République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderahmane Mira de Béjaia

Faculté des Sciences Exactes
Département d'Informatique
Ecole Doctorale Réseaux et Systèmes Distribués

Mémoire de Magistère
En Informatique

Option
Réseaux et Systèmes Distribués

Thème

Communication Protocols for Web Services

Présenté par

Mr. FARAH Zoubeyr

Devant le jury composé de :

Président :	Abdelnasser DAHMANI,	Professeur, Université de Bejaia, Algérie.
Rapporteur :	Zahir TARI,	Professeur, Institut (RMIT), Melbourne, Australie.
Examinateur :	Mohammed AHMED NACER,	Professeur, USTHB, Algérie.
Examinateur :	Nadjib BADACHE,	Professeur, USTHB, Algérie.
Invité :	Abdelkamel TARI,	Docteur, Université de Bejaia, Algérie.

Promotion 2007-2008

Dédicaces

Á ma mère

Á mon père

Á mes frères et sœurs

FARAH Zoubeyr

Remerciements

*Je tiens à remercier en premier lieu Monsieur **TARI Zahir** Professeur à l'institut RMIT, Melbourne, Australie, d'avoir accepté d'être mon encadreur durant cette année de magistère, et pour la confiance qu'il ma donnée et ses précieux conseils.*

*Je tiens à remercier vivement Monsieur **TARI Abdelkamel**, docteur à l'université de Bejaia et chef de département d'informatique et responsable de l'école doctorale d'informatique de Bejaia, pour les longues discutions, les conseils judicieux ainsi que les heures passées sur ce document.*

Je remercie très particulièrement M^elle^ ZEFOUNI Dalida, d'avoir accepté de lire ce travail et pour toutes ses remarques pertinentes.

Je remercie tous les étudiants de l'école doctorale ReSyD4, pour l'environnement de travail très agréable durant ces deux années de magistère.

Mes remerciements vont également aux membres de jury qui ont accepté de juger ce travail.

Résumé

Dans le domaine des Services Web les protocoles de communication peuvent être modélisés sous forme de communication d'automates d'états finis (CAEF). Bien que les automates d'états finis permettent une bonne modélisation de Services, ils n'offrent pas une vue de différents niveaux dans le cas de protocoles complexes. L'utilisation des modèles de communication d'automates d'états finis complexes (CAEFC) introduit de la modularité et de la hiérarchie sur la spécification de protocoles, permettant ainsi d'avoir une meilleure modélisation. Avant d'implémenter un protocole nous devons vérifier sa correction car une modélisation de protocole peut avoir quelques problèmes qui se traduisent par des erreurs de protocole, comme les erreurs d'interblocage et les transitions non exécutables. Donc, vérifier qu'un protocole ne contient pas des erreurs revient à valider sa spécification. L'opération de validation consiste à vérifier que tous les états du graphe d'accessibilité (d'exécution) de protocole ne présentent pas des erreurs logiques.

L'introduction des automates complexes nécessite la définition d'une technique de validation qui supporte de tel modèle et qui réduit la complexité de son analyse. Dans ce mémoire, nous proposons une technique de validation de protocoles modélisés sous forme de CAEFC, appelée RLRA (Reverse Leaping Reachability Analysis). Notre technique, permet la détection de toutes les erreurs de type interblocage dans une spécification de protocole. En se basant sur une approche de validation avec retour arrière, on commence par identifier de possibles états d'interblocage ensuite, valider parmi ces états suspects, ceux qui présentent réellement une erreur à travers la recherche des chemins de retours vers la racine du graphe de protocole. Pour réduire le problème de l'explosion combinatoire, nous introduisons les graphes de sauts (leap graphe) que nous exploitons dans la construction des chemins de retour arrière, ce qui apporte une réduction importante sur la taille du graphe à parcourir. Ce travail est terminé par un ensemble de tests et de comparaisons à fin de montrer l'efficacité de la technique proposée.

Mots clés : Services Web, protocole de communication dynamique, validation de protocoles, automate d'états complexes.

Abstract

In the Web services area, communication protocols can be modeled as communicating finite state machines (CFSM). Although finite state machines (FSM) allow good modeling of services, they do not offer a high level view in the case of complex protocols. The use of complex finite state machines adds modularity and hierarchy to the specification of protocols, and hence allows better modeling. Before implementing a protocol we need to verify its consistency. Modeling allows us to track protocol errors, such as deadlocks and non reachable transitions. Indeed, the validation of a protocol specification checks if it does not contain errors. The validation process consists of verifying all the execution states of the protocol by exploring its reachability graph.

The adoption of complex finite state machines requires a new validation technique that supports such a model and reduces its analysis complexity. Our work goes in this direction. In this thesis we propose a validation technique of protocols modeled with complex FSM, we called RLRA (Reverse Leaping Reachability Analysis). RLRA allows the detection of all deadlock errors in a protocol. Based on a validation with backtracking approach, it start by identifying a set of states that may have deadlocks before it determines which state from the suspected set have real error through backtracking to the root state of the protocol. To overcome the problem of combinatorial explosion, we use leap graphs to optimize backtracking and reduce considerably the size of the reachability graph. This work is concluded by a set of tests and comparisons that show the effectiveness of the technique.

Keywords : Web Services, dynamic communication protocol, protocols validation, complex finite state machines.

Sommaire

Liste des figures

Liste des tableaux

Abréviation

AAC : Analyse d'Accessibilité Compositionelle.
AEF: Automate d'États Finis.
AEFC : Automate d'États Finis Complexes.
CAEF : Communication d'Automate d'États Finis.
CAEFC : Communication d'Automate d'États Finis Complexes.
CPVA-ECSM: A Communication Protocol Validation Approach based on Exploration of Complex State Machines.
FRA: Fair Reachability Analysis.
FRG: Fair Reachability Graphe.
LRA :Leap Reachability Analysis.
PLEAP : Proper leap.
PROVAT : PROtocol Validation Testing.
RRA : Reverse Reachability Analysis.
RRA: Reverse Reachability Analysis.
RWSE: Random Walk State Exploration.
SRA: Simultaneouse Rechability Analysis.

Introduction générale

Le progrès du Web a permis la naissance d'une nouvelle génération de systèmes caractérisés par une meilleure intégration d'applications hétérogènes et une meilleure communication entre ses différents composants. Cela a fait surgir le besoin de permettre qu'une application client invoque un service d'une application serveur en utilisant Internet. Ce besoin a été l'origine de ce qu'on appelle service web. Un service Web est un composant logiciel qui offre des services à travers une interface standardisée [22]. La particularité des services Web réside dans le fait qu'ils utilisent la technologie Internet comme infrastructure pour la communication entre les composants logiciels (les services Web) et ceci en mettant en place un cadre de travail basé sur un ensemble de standards. Le cadre du travail des services Web peut être divisé en trois domaines : Un protocole de communication, une description de service et une découverte de service. Pour chacun de ces domaines est définit un standard : SOAP (Simple Object Access Protocol), qui permet la communication entre les services Web, UDDI (Universal Description, Discovery and Integration), qui est un annuaire de services Web et WSDL (Web Services Description Language), qui offre une description formelle des services Web.

Pour des composants logiciels présentant des services sous forme d'une collection d'unités de traitement (opérations) dont l'invocation n'excède pas un échange simple de messages (généralement requête-réponse), le paradigme des services Web est suffisant pour mettre en place des composants interopérables et facilement intégrables. Toutefois, certains composants logiciels nécessitent l'exécution d'une interaction plus complexe qui obéi à un protocole spécifique en vue de parvenir à la fonctionnalité attendue. Ce type de composant résulte d'une nécessité conceptuelle de la mise en place d'un service (composant agent) ou encore d'une agrégation modulaire de plusieurs services simples en services Web plus complexes. Dans un modèle de service web à base d'agents, un service web est assuré par un agent logiciel sur lequel est défini un ensemble d'opérations et un processus d'interaction, qui permettent d'avoir une invocation correcte du service. Pour fournir une interopérabilité complète entre agents sur Internet, il est non seulement exigé de ces agents la connaissance du format correct des données à échanger, mais également la connaissance des réponses valides, lorsque de multiples réponses sont possibles, ainsi que la phase d'initialisation et de terminaison de la conversation.

Une des solutions qui a été proposée pour palier à ce problème de conversation entre agents (services), est celle de conversation par automate d'états finis [38]. Dans cette approche, chaque

site serveur publie les détails permettant aux agents clients d'interagir avec lui. Cela implique la publication des spécifications de protocoles représentées par un automate d'états finis. Un agent client télécharge cette spécification, la valide pour sa correction et implémente le protocole dynamiquement. Cela peut être vue comme une négociation de protocoles où le client négocie pour implémenter toutes les exigences d'un serveur. Une spécification de protocole sous forme d'une communication d'automates d'états finis, modélise le comportement du service et le comportement requis par l'utilisateur pour avoir une bonne invocation du service. Dans ces conditions, un agent ne se soucierait pas du (des) standard(s) implémenté(s) par un autre agent pourvu qu'il existe une compréhension de la communication requise.

Quand un agent client télécharge une spécification de protocole (sous forme d'une communication d'automates d'états finis) afin de communiquer avec un autre agent sur Internet, il a besoin de vérifier sa correction car un automate d'états finis peut avoir quelques problèmes qui se traduisent par des erreurs de protocole, comme les erreurs d'interblocage et les transitions non exécutables. Donc, vérifier qu'un protocole ne contient pas des erreurs revient à valider sa spécification.

Il existe plusieurs techniques qui permettent la validation de protocole. Les plus connues sont l'analyse d'accessibilité standard et l'analyse d'accessibilité réversible. L'analyse d'accessibilité standard a été la technique la plus utilisée dans le domaine de validation de protocole. Le principe de cette technique est l'énumération de toutes les interactions possibles entre les entités communicantes du protocole. Cette énumération génère des états globaux de protocole où chaque état global donne la situation du protocole après avoir exécuté une opération. A la fin de cette opération, on obtient un graphe d'accessibilité de protocole. Les erreurs de protocole, telles que les interblocages et les réceptions non spécifiées, sont alors détectées par la vérification de chaque état global composant ce graphe.
L'analyse d'accessibilité standard est une technique intuitive, simple et permet la détection de toutes les erreurs que peut contenir une spécification de protocole. Cependant, elle soufre du problème de *l'explosion combinatoire*. En général, le nombre total des états globaux d'un protocole est exponentiel en terme de nombre d'entité composant le protocole ainsi que le nombre d'état local de chaque entité [35]. Bien que plusieurs versions (améliorations) ont étés proposées [1,8,18,5] pour allégir l'analyse d'accessibilité standard, le problème de l'explosion combinatoire était toujours présent, ce qui a limité leur application à des protocoles simples.
L'analyse d'accessibilité réversible [14] est une alternative intéressante de l'analyse d'accessibilité standard. A partir des propriétés qui caractérisent les erreurs de type interblocage, on dégage un ensemble d'états globaux qui vérifient ces propriétés. Ces états sont appelés états suspects. Par la suite, un état suspect est confirmé comme une erreur d'interblocage si l'état global initial du protocole est atteignable par une opération de retour arrière à partir de cet état

suspect, sinon l'état suspect est retiré de l'ensemble des erreurs de protocole. Avec ce procédé, cette technique permet de parcourir seulement un sous-ensemble de l'espace des états du protocole. Cette technique permet la détection de toutes les erreurs d'interblocage dans une spécification de protocole.

Les deux familles de techniques citées ci-dessus ont été proposées pour valider des spécifications de protocole utilisant un modèle de communication d'automate d'états finis simple [19]. Toute fois, dans un contexte de service web, nous avons besoin d'avoir un modèle de spécification qui permet de décrire clairement le service offert. Cela ne peut être obtenu que par la prise en compte de la modularité et la hiérarchie des services Web.
L'utilisation des automates d'états finis complexes [30] permet d'introduire de la modularité et de la hiérarchie sur les spécifications de protocole. Toute fois, valider de telles spécifications nécessite une technique de validation qui prend en compte les nouvelles propriétés de ces modèles de spécification complexe.

L'objectif de notre travail entre dans ce cadre, qui consiste à proposer une technique de validation de protocoles modélisés sous forme de communication d'automates d'états finis complexes. Pour cela, nous proposons une nouvelle technique de validation de protocole basée sur le principe de retour arrière. L'avantage que présente notre technique par rapport à la technique de validation avec retour arrière simple est qu'elle est applicable sur des modèles de CAEF simple et des modèles de CAEF complexe. Un deuxième avantage, concerne l'opération de retour arrière, où nous exploitons les graphes avec saut [17] pour effectuer un retour arrière optimisé. Les graphes avec saut permettent l'exploration d'un minimum d'état global dans un graphe d'accessibilité de protocole, tout en gardant la capacité d'atteindre les états qui présentent des erreurs de types interblocage. Cela permet de réduire considérablement le nombre d'état à vérifier pendant l'analyse du protocole.

Ce mémoire est organisé comme suite : dans le premier chapitre, nous donnons une introduction sur les services web, leur architecture ainsi que leur fonctionnement. Dans le deuxième chapitre, nous présentons le protocole de communication dynamique entre services Web DYNWES, les différentes notions sur les automates d'états finis sont aussi présentées dans ce chapitre. Dans le troisième chapitre, nous résumons un ensemble de techniques de validation de protocoles sous forme d'un état de l'art. Notre contribution vient dans le chapitre quatre. Nous donnons toutes les définitions et principes utilisés par notre technique ainsi que les preuves formelles de son applicabilité. Pour montrer l'efficacité de notre technique, nous l'avons implémenté et exécuté sur un ensemble d'exemples de tests et de comparaisons. Les différents résultats obtenus sont donnés dans le chapitre cinq. Nous terminons ce mémoire par une conclusion et des perspectives futures de notre travail.

Chapitre I

Introduction aux Services Web

I-1) Introduction

Les services Web sont un paradigme naissant qui vise à la transposition des architectures par composant dans le cadre du Web.
Les services Web se basent sur le standard XML (*eXtensible Markup Language*) pour la description et l'échange de données. Ils séparent la description des services de leurs implémentations en définissant des interfaces dans le langage WSDL (*Web Services Description Language*) et les publient dans le référentiel UDDI (*Universal Description, Discovery and Integration*) et communiquent en utilisant le protocole standard SOAP (*Simple Object Access Protocol*), permettant ainsi, d'obtenir une architecture faiblement couplée et interopérable.

Dans ce chapitre, nous définissons les services Web, leur architecture de base ainsi que leur principe de fonctionnement. Nous définissons aussi les principaux standards utilisés par les services Web (SOAP, WSDL et UDDI).

I-2) Historique des services Web

Les services Web s'agissent d'une technologie jeune ; Née à la fin des années quatre-vingt-dix, sous l'impulsion de géants de l'informatique comme Microsoft, IBM[1], Sun[2]. Ils prennent leur origine dans l'informatique distribuée et dans l'avènement du Web. Le but de l'informatique distribuée est de permettre à une application sur une machine d'accéder à une fonction d'une autre application sur une machine distante et ce, de la même manière que l'appel d'une fonction locale, indépendamment des plates formes et des langages utilisés. Trois standards sont aujourd'hui utilisés : CORBA/IIOP (*Common Object Request Broker Architecture/ Internet Inter-ORB Protocol*), DCOM (distributed Component Object Model) et RMI (*Remote Method Invocation*). Ces modèles offrent des mécanismes de récupération d'espace mémoire,de sécurité et de gestion du cycle de vie des objets, mais ils sont complexes, peu compatibles avec les pare-feux et difficilement interopérables entre eux. Ils restent donc souvent confinés à l'intérieur des entreprises.
Le second point qui a favorisé l'essor des services Web, est le développement d'Internet par la démocratisation du haut débit, structuration des données via XML et la recherche d'interopérabilité. Les entreprises publiaient déjà de l'information via des sites web, utilisaient la messagerie et faisaient du commerce électronique, elles l'utilisent maintenant pour leurs applications métiers (ressources humaines, ventes, finances, etc.).
La première société à avoir introduit un concept de services distribués, proche de ce qui existe aujourd'hui, est Hewelett-Packard avec son produit e-Speak, et ce dès 1999. La suite fut une grande course entre les principaux acteurs du marché qui ont progressivement, par soucis d'interopérabilité, adopté les principaux standards des services Web qui sont SOAP,WSDL et UDDI [23].

I-3) Définition des services Web

Le consortium **W3C**[3] définit un service Web par :
« A Web service is a software application identified by a URI[4], whose interfaces and binding are capable of being defined, described and discovered by XML artefacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols ».
Un service Web est une application logicielle identifiée par un URI dont les interfaces et les liaisons sont définies, décrites et découvertes en XML. Cette application supporte une interaction

[1] International Business Machines
[2] Stanford University Network
[3] W3C: World Wide Web Consortium. Organisme de normalisation des standards du Web. *http://www.w3.org*
[4] URI : Uniform Ressource Identifier.

directe avec d'autres applications logicielles en utilisant des messages XML via des protocoles Internet.

IBM donne dans un tutoriel la définition suivante des services Web [34] :
« Les web services sont la nouvelle vague des applications web. Ce sont des applications modulaires, auto-contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le web. Les web services effectuent des actions allant de simples requêtes à des processus métiers complexes. Une fois qu'un web service est déployé, d'autres applications (y compris des web services) peuvent le découvrir et l'invoquer ».

L'objectif ultime de l'approche Services Web est de transformer le Web en un dispositif distribué de calcul où les programmes (services) peuvent interagir de manière intelligente en étant capables de se découvrir automatiquement, de négocier entre eux et de se composer en des services plus complexes [26].

I-4) Architecture et infrastructure des services Web

L'architecture des services Web est une architecture orientée service (SOA Service Oriented Architecture) [7]. L'architecture SOA est un modèle qui définit un système par un ensemble de composants logiciels distribués qui fonctionnent en concert afin de réaliser une fonctionnalité globale préalablement établit [39]. Les composants dans un système distribué n'opèrent pas dans un même environnement de traitement et sont obligés de communiquer par échanges de messages afin de solliciter des services dans le but d'accomplir le résultat souhaité.

La définition de l'architecture des services Web consiste à mettre en évidence les concepts, les relations entre ces concepts ainsi qu'un ensemble de contraintes entre ces concepts [22].

Les principaux concepts intervenant dans l'architecture des services Web sont [9]:
– le fournisseur de services : désigne le serveur qui héberge les services déployés ;
– le client du service : représente l'application cliente qui invoque le service ;
– le service : désigne les fonctionnalités d'un agent logiciel qui implémente le service ;
– la description du service : c'est la spécification du service exprimée dans un langage de description interprétable par les machines, c'est-à-dire une description technique dans laquelle le service est vu en termes de messages, de types, de protocoles de communication et d'une adresse physique ;
– les messages : c'est la plus petite unité d'échange entre les clients et les services. La structure des messages qui permettent l'invocation d'un service doit être exprimée dans la description du service ;

– la ressource : désigne l'identifiant du service, c'est-à-dire son URI.

Bien que le Web soit constitué de plates-formes totalement hétérogènes où les intérêts des différents acteurs du marché s'entremêlent, ceci ne l'a pas empêché de se développer et d'être universel. Ce succès est du essentiellement à l'édition d'un ensemble de standards ouverts, dont les plus connus sont le protocole http (HyperText Transfer Protocol) et le format MIME (Multipurpose Internet Mail Extension). Ensemble, ils offrent un mécanisme d'échange de données de toutes sortes quelle que soit la nature des plates-formes impliquées. Le développement des services Web a suivi la même approche en mettant en place une campagne de standardisation qui a touché les aspects les plus importants du développement d'un modèle d'intégration d'applications hétérogènes.
L'avantage de ce modèle est de présenter ces services comme des « boıtes noires ». En fait, les entrées-sorties d'un service sont gérées au sein de messages dont on connaît le format grâce à des interfaces clairement exposées mais sur lesquelles l'implémentation interne du traitement n'influe pas au niveau de la structure. Ceci permet un haut niveau de modularité et d'interopérabilité. L'avantage du modèle de message est qu'il permet de s'abstraire de l'architecture, du langage ou encore de la plate-forme qui va supporter le service : il suffit juste que le message respecte une structure donnée pour qu'il puisse être utilisé.

L'originalité de l'infrastructure des services Web consiste à mettre en place ces services exclusivement sur la base des protocoles les plus répandus sur Internet. Ces protocoles sont repartis selon quatre axes représentés sur la figure 1.1 suivante:

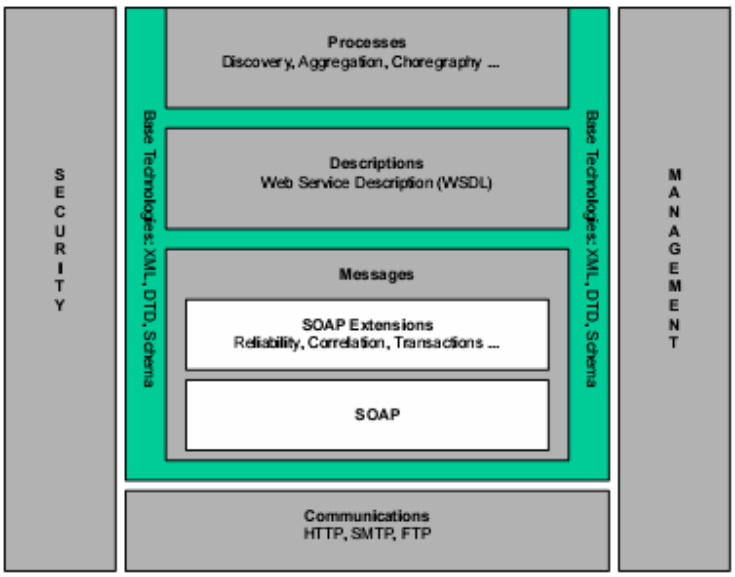


Figure 1.1 : Architecture des services Web.

– Couche de transport : cette couche s'occupe de transporter les messages entre applications en utilisant les protocoles HTTP, FTP ou SMTP;
– Messages XML : il s'agit de formaliser les messages à l'aide d'un vocabulaire XML commun (SOAP) ;
– Description de services : description de l'interface publique des services Web (WSDL) ;
– Recherche de services : centralisation des services et de leur description dans un référentiel commun (UDDI).

I-5) Déploiement, recherche et invocation de services Web

Le cycle de vie d'un service Web se déroule de la façon suivante : une fois créé, le service est déployé sur le réseau (local ou Internet). Puis un utilisateur ayant des besoins spécifiques va rechercher un service correspondant à ses besoins à l'aide d'un annuaire spécialisé. Enfin, une fois le service trouvé, l'utilisateur va invoquer le service : une communication va être mise en place entre l'utilisateur et le service Web.
Ce cycle de vie, représenté par la figure 1.2 [22], fait appel aux trois grandes technologies : SOAP, WSDL et UDDI.

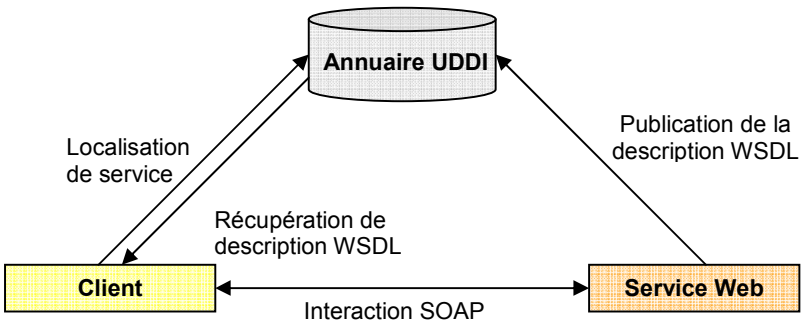


Figure 1.2 : Déploiement, découverte et invocation de services Web

Protocole SOAP : Le protocole SOAP (Simple Object Access Protocol), est un protocole de communication basé sur XML qui permet aux services Web d'échanger des informations dans un environnement décentralisé et distribué tout en s'affranchissant des plates-formes et des langages de programmation utilisés. Il définit un ensemble de règles pour structurer les messages envoyés. Mais SOAP ne fournit aucune instruction sur la façon d'accéder aux services Web. C'est le rôle du langage WSDL.

Description WSDL : La spécification WSDL joue un rôle important dans l'interopérabilité des composants services Web. Moyennant un schéma uniforme obéissant à une sémantique bien définie (figure 1.3), elle permet aux composants de définir ce qui est nécessaire à leur invocation. La spécification WSDL est définie selon une sémantique totalement indépendante du modèle de programmation de l'application.
Elle sépare clairement la définition abstraite du service (échange de messages) de ses mécanismes de liaison (définition des protocoles applicatifs). Cette dernière caractéristique permet aux composants d'interagir même si l'application a été modifiée, ce qui est un point important pour assurer l'interopérabilité des services.

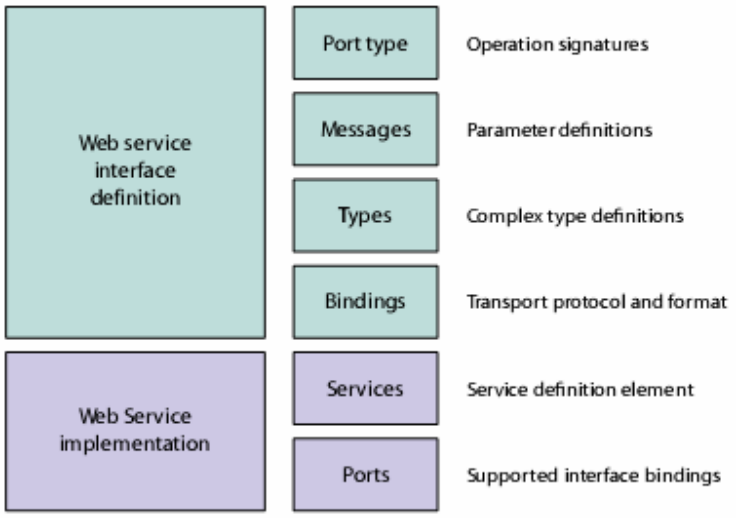


Figure 1.3 : Architecture WSDL.

Référentiel UDDI : Dans un environnement ouvert comme Internet, la description d'un service Web n'est d'aucune utilité s'il n'existe pas de moyen de localiser aussi bien les services Web que leurs descriptions WSDL : c'est le rôle des référentiels UDDI (Universal Description, Discovery and Integration). La spécification UDDI constitue une norme pour les annuaires de services Web. Les fournisseurs disposent d'un schéma de description permettant de publier des données concernant leurs activités, la liste des services qu'ils offrent et les détails techniques de chaque service. La spécification offre également une API (Application Programming Interface) aux applications clientes, pour consulter et extraire des données concernant un service et/ou son fournisseur.

I-6) Conclusion

Les services Web permettent à des applications de dialoguer via Internet par échange de messages, indépendamment des plates formes et des langages sur les quels sont implémentées en s'appuyant sur un ensemble de standards (XML, SOAP, WSDL, UDDI).

Les standards actuellement adoptés par les services web permettent de garantir l'interopérabilité entre services. Cependant, ils soufrent d'une profonde limitation lorsque on traite l'aspect dynamique des services Web (conversation dynamique, sécurité et composition de services Web).

Le chapitre suivant, décrit un modèle de communication dynamique pour les services web à basent d'agents afin de permettre une interaction dynamique entre ces derniers.

Chapitre II

Un Protocole de communication dynamique pour les services Web (DynWes)

II-1) Introduction

Les services Web, tels qu'ils sont définis actuellement, sont limités à des fonctionnalités relativement simples. Toutefois, pour certains types d'applications, nous avons besoin de définir un processus d'interaction plus complexe entre services afin d'atteindre la fonctionnalité attendue. Ce besoin peut se présenter dans l'un des deux cas suivants :

- Le premier cas est rencontré lorsque un service Web est implémenté à base d'agents, dans ce cas la, il est composé d'un ensemble d'opérations accessibles et d'un modèle d'interaction qui permet de définir l'enchaînement de ces différentes opérations à fin d'utiliser correctement le service.

- Le deuxième cas se présente lorsqu'il est nécessaire de combiner un ensemble de services Web « basiques » en un service plus « sophistiqué » afin de répondre à des exigences plus complexes : c'est l'agrégation de services web.

Dans ce chapitre, Nous présentons un protocole dynamique pour les services Web à base d'agent [38] qui définit une interaction dynamique entre services Web en se basant sur un modèle de spécification par automate d'états finis. Les automates d'états finis, la communication d'automates d'états finis et la validation de protocoles sont aussi des notions que nous détaillons dans ce chapitre.

II-2) DynWes, Protocole de communication dynamique pour les services Web

Tari et al[31, 32, 38] proposent un modèle de communication dynamique pour les services web à base d'agents. Dans cette approche, Chaque site serveur publie les détails permettant aux agents clients d'interagir avec lui. Cela implique la publication des spécifications de protocoles représentées sous forme d'une communication d'automates d'états finis(CAEF). Un agent client télécharge cette spécification, la valide pour sa correction et implémente le protocole dynamiquement. Cela peut être vue comme une négociation de protocoles où le client négocie pour implémenter toutes les exigences d'un serveur.

Les avantages d'implémenter un protocole de conversation dynamiquement sont:

1. La spécification du protocole est séparée de son code compilé car chaque client utilise son propre code d'implémentation.
2. L'interopérabilité à la demande est assurée et par conséquent ceci offre une interaction possible avec un très grand nombre de services sur Internet.

DynWES implémente un service web particulier en utilisant un agent (écrit dans un langage de programmation) et un agent différent le lendemain (écrit probablement dans un langage de programmation différent). Les agents échangent les données dans le format XML (c-à-d, automates d'états téléchargés) et communiquent avec des opérations bien définies.

II-3) Conversation juste à temps

En utilisant une approche de conversation dynamique, nous pouvons modéliser l'interopérabilité entre agents comme une communication entre automates. Si le processus d'un agent serveur est modélisé sous forme d'un automate d'états finis (AEF) alors cette spécification doit être disponible à l'agent client. Cette spécification décrit le service fournit et le comportement que dois avoir un client désirant utiliser ce service. Ceci signifie qu'un agent client peut interagir avec un agent serveur sans connaissance initiale du niveau de conversation du protocole.

L'approche client/serveur est plus simple pour implémenter dynamiquement et juste à temps (JIT[5]) les protocoles de conversation car un client a besoin seulement d'apprendre du protocole exigé par le serveur, ou de multiples serveurs. Bien sûr, Dans un environnement ouvert (Internet), il est possible qu'une relation << many-to-many >> soit requise pour l'implémentation dynamique du protocole.

[5] Just-In-Time

La figure2.1 décrit un client qui télécharge une spécification et qui converse avec un agent marchand en utilisant un AEF.

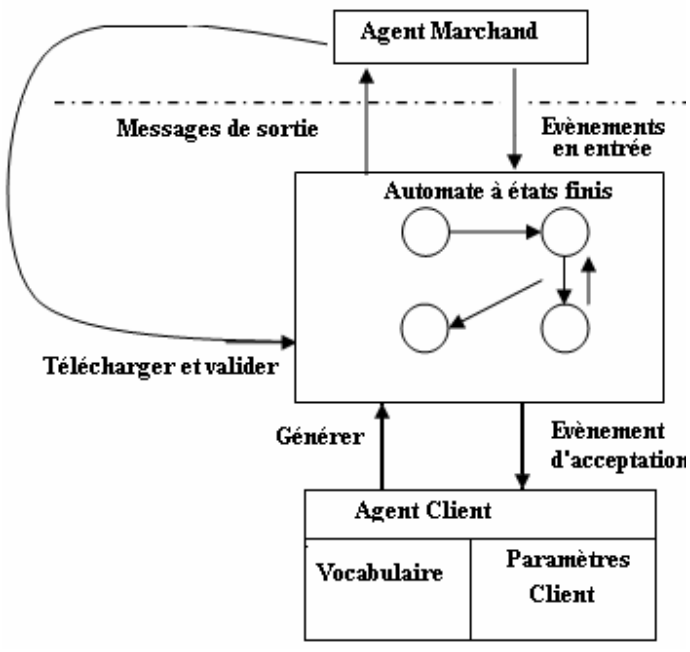


Figure 2.1 : Exemple d'un automate d'états juste à temps[38].

Lorsqu'un service d'un autre marchant est exigé, l'agent client peut télécharger une autre spécification sous forme d'automate d'états et répéter le processus. Potentiellement, l'agent client peut converser maintenant avec n'importe quel nombre d'agents marchands qui fournissent leurs spécifications sous forme d'AEF pour les clients.

II-4) Modélisation de protocoles sous forme d'AEF

Il existe plusieurs modèles de spécification de protocoles, le modèle de Communication d'Automate d'Etats Finis (CAEF) [19] et les réseaux de Petri[40] sont les plus utilisés. Ces deux modèles présentent l'avantage d'être simples d'utilisation et permettent le développement de méthodes de validations automatisables. Dans ce projet nous nous intéressons aux modèles de CAEF.

II-4-1) Modèle de Communication d'Automates d'Etats Finis (CAEF)

Le modèle de CAEF permet la représentation d'un protocole de communication, les entités impliquées dans cette communication ainsi que les différents canaux qui relient ces entités entre elles. Ce modèle de représentation rend l'analyse des protocoles plus facile et très complète, en plus il facilite la détection de tout type d'erreur de conception que peut contenir une spécification de protocoles.

Le modèle de CAEF modélise une communication entres des entités dont le comportement est spécifié sous forme d'un Automate d'Etat Fini (AEF). Un AEF est un ensemble d'états, et un ensemble de transitions définies sur ces états. Un état représente la situation d'une entité à un instant donné de la communication. A un instant donné, une entité du protocole ne peut être en deux états différents. Une entité exécute une transition et passe d'un état vers un autre lors de l'envoi ou de la réception d'un message.

Formellement, un AEF peut être représenté par un quintuple $(\mathbf{S,S_0,S_f,A,\Delta})$ tel que :

- **S** : ensemble des états de l'automate.
- $\mathbf{S_0}$: état initial de l'automate $\mathbf{S_0 \in S}$.
- $\mathbf{S_f}$: ensemble des états finaux $\mathbf{S_f \subset S}$.
- **A** : alphabet de l'automate, elle représente les messages qui peuvent être utilisés dans la conversation.
- **Δ** : l'ensemble des transitions de l'automate.

  La relation de transition peut être représentée par un quadruplet **(d,f,m,e)** tel que :
  - **d∈S** représente l'état de départ de la transition.
  - **f∈S** l'état où se termine la transition.
  - **m∈A** représente le message envoyé ou reçu.
- **e** représente la nature de l'événement de la transition ((-) émission et (+) réception).

Un exemple d'une modélisation sous forme d'AEF est donné sur la figure2.2. Le processus modélisé est une opération de réservation de billet d'avion en ligne. L'ensemble des états de cet automate **S,** est composé des états *IDLE*, *Reservation*, *Formulaire*, *Valid.Res*, *Confirmation*, *Val.Final* et *Ebillet*. L'état initial $\mathbf{S_0}$ de l'automate est *IDLE*. L'ensemble des états finaux $\mathbf{S_f}$ contient le seul état *Ebillet*. L'alphabet **A** de cet automate est constitué des messages *Réserver, Envoyer Form, ValiderRes, ConfirmerRes, Virement, ModifierAnnuler* et *Contrat*.

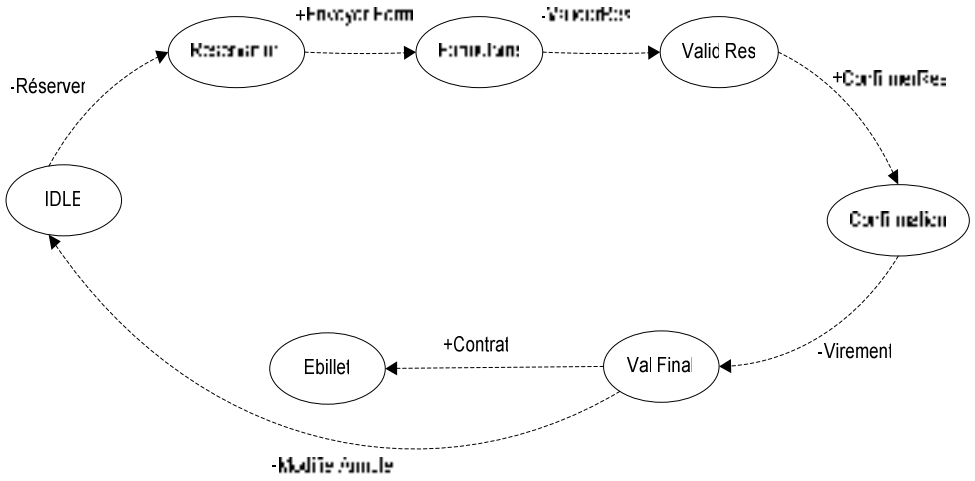


Figure 2.2 : AEF modélisant un service de réservation de billet d'avion.

Le processus est initialement à l'état *IDLE*, il passe à l'état *Réservation* lors de la réception d'un message de demande de réservation (*Réserver*). Un formulaire doit être ensuite rempli et validé pour valider la réservation et enfin la confirmée par un virement bancaire. Une fois le virement est accompli un contrat est envoyé au client.

Canaux de communication

Les différentes entités d'un modèle de CAEF échangent des messages via des canaux de communication. Un message envoyé par une entité de ce modèle doit être sauvegardé pour qu'il soit reçu par une autre entité. Les canaux de communication sont généralement des files de capacité limitée avec un ordre FIFO.

Dans l'exemple de la figure2.3. Les entités M1 et M2 communiquent à travers deux canaux C_{12} et C_{21}. L'entité M1 reçoit les messages envoyés par M2 sur le canal C_{21} et envoie des messages vers M2 sur le canal C_{12}. L'entité M2 utilise le canal C_{21} pour envoyer des messages vers M1 et reçoit les messages en provenance de M1 sur le canal C_{12}.

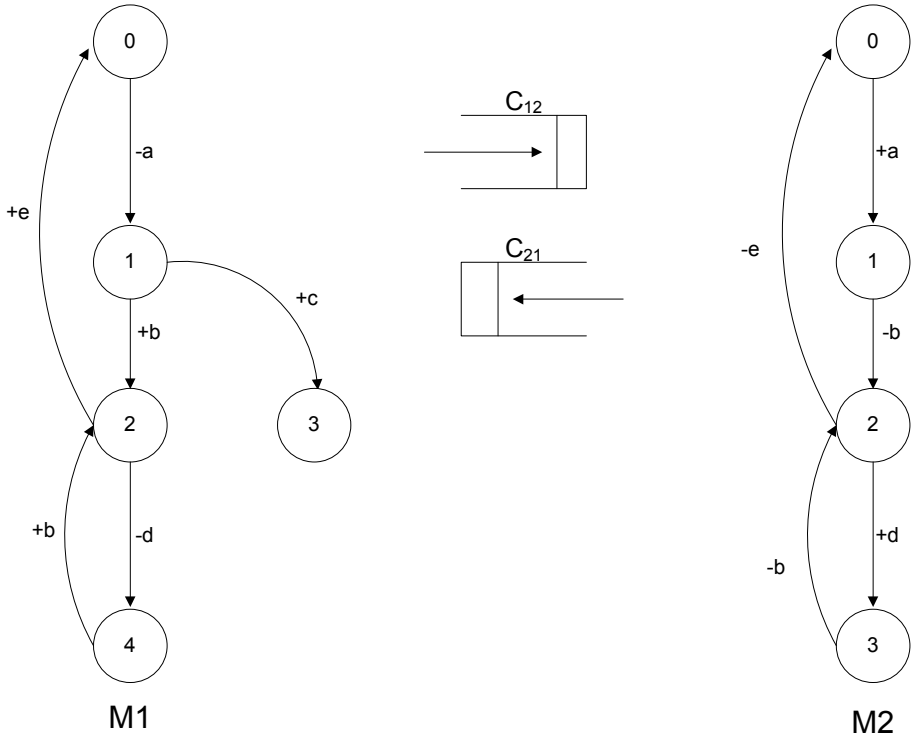


Figure 2.3 : Communication entre deux AEF.

Initialement les deux entités sont dans leurs états initiaux et leurs canaux de communication sont vides. Lorsque M1 envoie un message 'a' et passe à l'état 1, le contenu du canal C_{12} devient 'a'. Après cet envoi, M2 peut changer d'état et passer à l'état 1 après avoir reçu le message 'a' sur le canal C_{12}. M2 envoie un 'b' et passe à l'état 2 et le message 'b' est mis dans le canal C_{21}, donc M1 peut le recevoir et passer à l'état 2. Ainsi la communication continue entre M1 et M2 jusqu'à ce que les deux entités arrivent dans leurs états finaux.

Etat global d'un protocole

L'état global d'un protocole de communication est représenté par l'état, à un instant donné, de toutes les entités impliquées dans la communication ainsi que le contenu des différents canaux de communication reliant ces entités entre elles. Soit n>=2 le nombre d'entité communicantes d'un protocole de communication. Si I={1,2…n} représente l'indice de l'ensemble des entités du protocole, alors l'état global de ce protocole sera représenté par le couple <S,C>, où $S=\{s_i/i\in I\}$ est l'ensemble de toutes les entités impliquées dans la communication et $C=\{c_{ij}/(i,j)\in I\times I \text{ et } i\neq j\}$ est l'ensemble de tous les canaux définis pour la communication, tel que c_{ij} représente le canal de communication reliant l'entité i à l'entité j.

Pour l'exemple de la figure2.3 précédente, l'état global du protocole est représenté par le vecteur $<S_1, S2, C_{12}, C_{21}>$ tel que S_1 et S_2 représentent respectivement les états locaux des entités M_1 et

M_2. Par exemple, l'état global initial du système est donné par <0,0,∅,∅>, dans cet état global les deux entités sont dans leurs états initiaux et le contenu de leurs canaux est vide.

II-4-2) Modèle de Communication d'Automates d'Etat Fini Complexe (CAEFC)[30]

Un modèle de communication d'Automate d'États Finis Complexes (AEFC) introduit la notion d'état complexe, qui est un AEF dont certains de ses états sont eux-mêmes des AEF (internes). Cela permet d'introduire de la modularité et de la hiérarchie sur les spécifications par AEF. Les états qui rentrent dans la composition d'un état complexe sont appelés **états internes.** Le développement des différents AEF internes dans une spécification d'AEFC donne un AEF aplati, qui est un AEF ordinaire.

Cette représentation de comportement permet d'avoir une conversation entre agent sur plusieurs niveaux. Chaque niveau de conversation est spécifié par un AEF. Cela permet d'effectuer une validation en multi niveaux où chaque niveau de conversation est validé indépendamment des autres niveaux.

Formellement, un AEFC peut être représenté par un quintuple (S,S_0,S_f,A,Δ) tel que :

- **S** : ensemble des états de l'automate, il comprend des états simples et des états complexes. Chaque état complexe doit être défini par un AEF.
- S_0 : état initial de l'automate $S_0 \in S$.
- S_f : ensemble des états finaux $S_f \subset S$.
- **A** : alphabet de l'automate, elle représente les messages qui peuvent être utilisé dans la conversation.
- **Δ** : les transitions de l'automate.

  La relation de transition peut être représenté par un quadruplet **(d,f,m,e)** tel que :
  - **d∈S** représente l'état de départ de la transition.
  - **f∈S** l'état où se termine la transition.
- **m∈A** représente le message envoyé ou reçu.
- **e** représente la nature de l'événement de la transition (émission ou réception).

Dans le cas ou **f** est un état complexe (représente un AEF interne), le résultat de la transition va mettre l'automate interne sur sont état initiale, si **d** et aussi un état complexe, la transition ne peut être exécutée que si l'Automate qui lui correspond atteint sont état final.

Un exemple de spécification sous forme d'AEFC est donné sur la figure2.4. L'exemple modélise un service de réservation de billets d'avion en ligne (côté serveur).

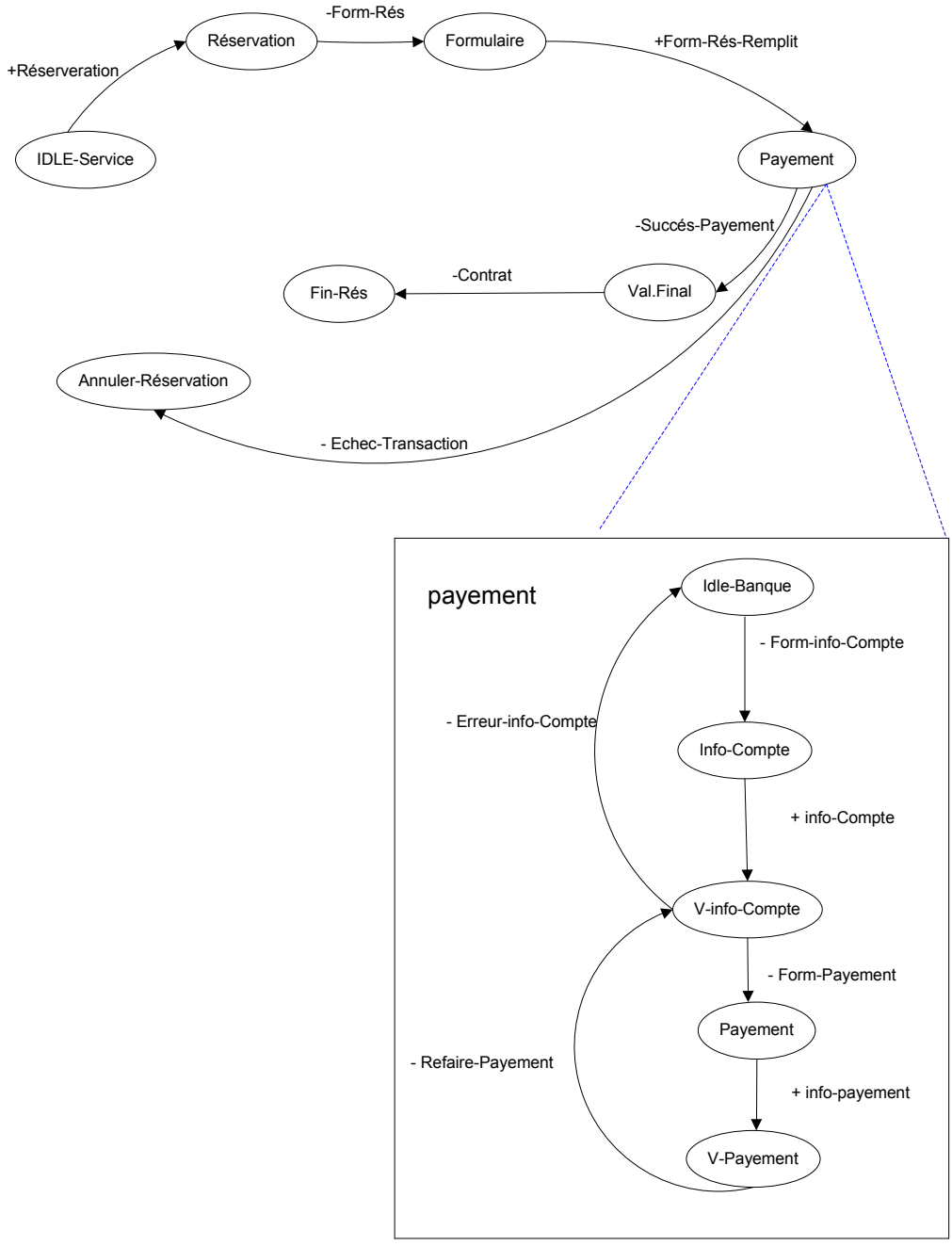


Figure2.4 : Exemple de spécification sous forme d'AEFC

Un modèle de CAEFC est constitué de deux ou plusieurs entités, modélisées sous forme d'AEFC. L'échangent de messages entre les entités de ce modèle se fait via des canaux de communication FIFO bornés.

La figure2.5 donne la spécification du coté client correspondant au service de réservation de billet d'avion sur la figure2.4 précédente. Ces deux entités (Fig2.4 et Fig2.5) communiquent par échange de messages via deux canaux de communications FIFO unidirectionnels. Le premier canal permet l'envoi de messages de l'agent fournisseur de service vers le client et le deuxième canal permet au client d'envoyer des messages vers l'agent fournisseur de service.

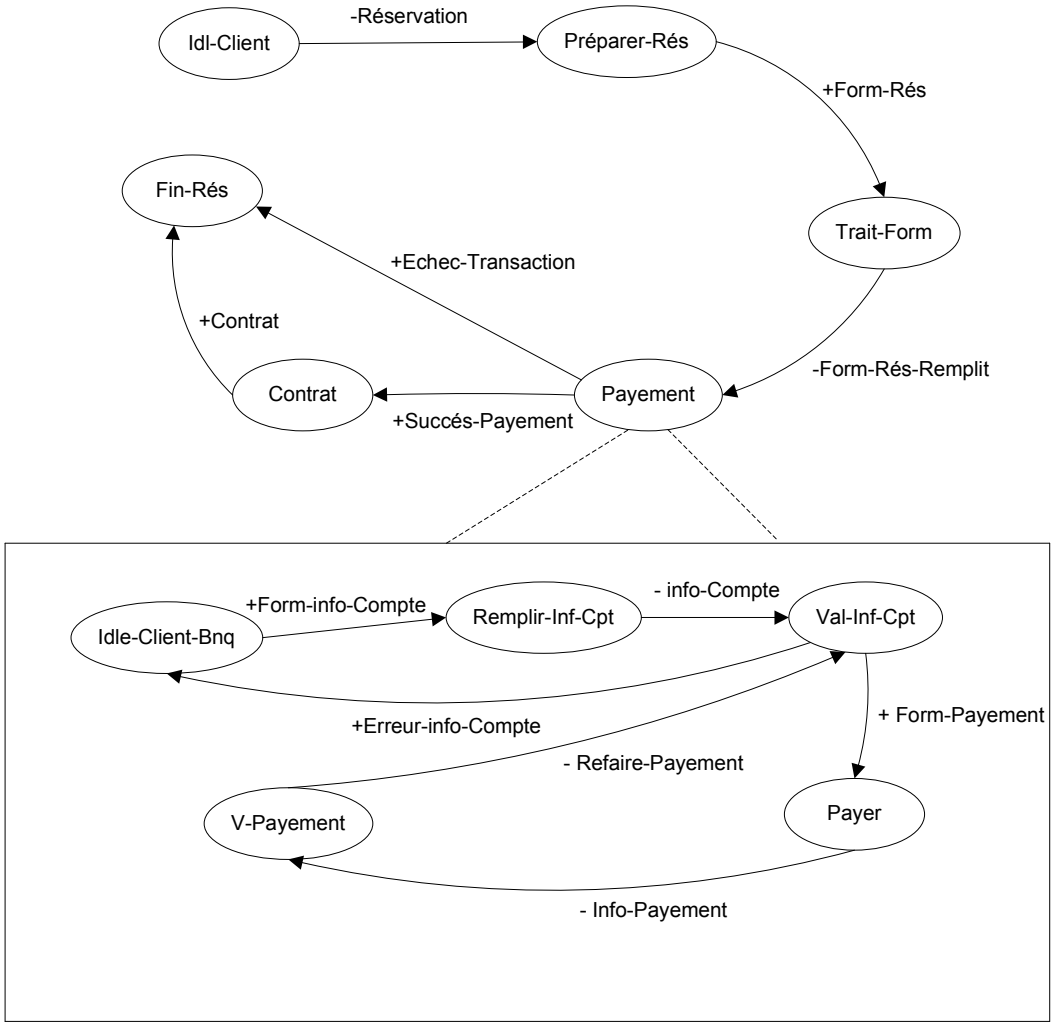


Figure2.5 : spécification sous forme d'AEFC de l'agent client

II-5) Erreurs logiques de protocole

Les erreurs logiques de protocole correspondent à des états puits dans la spécification du protocole. Ces erreurs peuvent être de différentes natures.

II-5-1) Différentes erreurs logiques de protocole

Les erreurs logiques que peut contenir une spécification par AEF sont les suivantes :

Interblocage : une erreur de type interblocage apparaît lorsque toutes les entités du protocole sont en état d'attente de réception de message et leurs canaux d'entrée sont vides. Dans ce cas la, aucune entité ne peut recevoir le message qu'elle attend est donc aucun message ne peut être échangé dans le système, ce qui présente une situation de blocage.

Réceptions non spécifiées : on dit qu'une erreur est de type réception non spécifiée lorsque un état reçoit un message en entrée et qu'aucune opération n'est spécifiée pour son traitement.

Canal débordé : un canal est dit débordé, lorsque il contient un nombre de messages égale à sa capacité et que d'autres entité envoie encore d'autre messages sur ce canal.

Boucle infinie (Livelock) : une situation de livelock se présente lorsque deux ou plusieurs entités du protocole continuent d'échanger les mêmes messages et qu'il n'y est aucune progression dans l'exécution du protocole.

Transitions non exécutables : une transition non exécutable provient généralement des erreurs de programmation.

II-5-2) Types d'erreur dans une spécification par AEFC

En plus de la nature de l'erreur logique, l'utilisation des automates d'états complexes nécessite la définition du type de l'erreur détectée qui peut être de type simple, complexe ou hybride selon la nature des états qui ont provoqués l'erreur :

Erreurs logiques simples

Ce type d'erreur apparaît dans le cas où chaque entité communicante du modèle se trouve dans un de ses états simples. L'exemple de la figure2.6 contient une erreur de ce type. La spécification de cette exemple modélise une communication entre deux entité M1 et M2. Initialement les deux entités se trouvent dans leurs états initiaux ce qui correspond à l'état global du système <0,0,ε,ε>. M1 commence la communication en envoyant le message 'a' sur le canal de communication C12 et passe à l'état 1. M2 récupère le message 'a' envoyé par M1 depuis le canal C12 est passe à l'état 1. En examinant l'état global du modèle résultant des deux exécutions precedentes, qui est <1,1,ε,ε>, l'entité M1 est en attente de réception d'un message 'b' ou 'c' pour pouvoir continuer son exécution, de même l'entité M2 est en attente de réception d'un

message ‘b’ pour pouvoir avancer. Dans cet état global on identifie une erreur d’interblocage **simple** car les deux entités se trouvent dans des états simples.

Erreurs logiques complexes

Ce type d’erreur apparaît seulement dans des modélisations d’AEFC. Lorsque les entités communicantes du système se trouvent dans un de leurs états complexes et une erreur est détectée à cet état global, on dit que l’erreur est de type complexe. L’exemple de communication d’AEFC de la figure 2.6 contient une erreur logique d’interblocage complexe à l’état global <5,5,ε,ε>. Dans cet état les entités M1 et M2 se trouvent dans leurs états complexes S1 et S1 respectivement.

Erreurs logiques hybrides

Ce type d’erreur est rencontré lorsque une ou plusieurs entités du modèle se trouvent dans un état simple et le reste des entités se trouvent dans des états complexes. Dans l’exemple de la figure2.6, lorsque M1 est dans l’état 6 de son état complexe S2 et l’entité M2 est dans son état simple 6, une erreur de type interblocage hybride est détectée dans cet état du protocole (6,6,ε,ε).

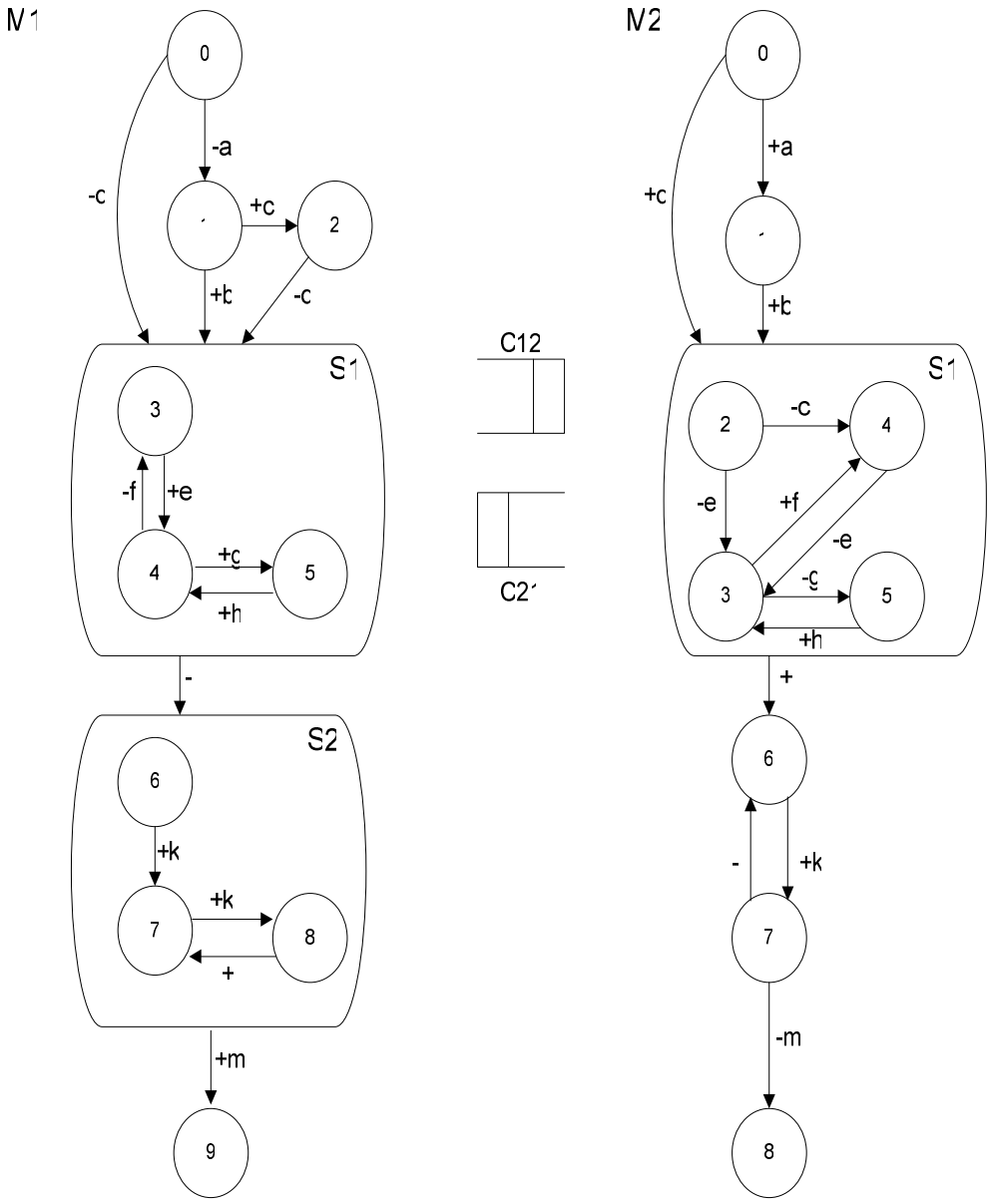


Figure2.6 : spécification d'une communication d'AEFC.

II-6) Validation de protocole

Quand un client télécharge une spécification sous forme d'AEF afin de communiquer avec un autre agent sur Internet, il a besoin de vérifier sa correction car un AEF peut avoir quelques problèmes qui se traduisent par des erreurs de protocole (interblocage, livelocks …etc). Pour effectuer cette vérification, un client doit implémenter une technique de validation de protocole (Chapitre 4). La validation d'un protocole revient à faire une analyse de son graphe d'accessibilité. Un graphe d'accessibilité donne les différentes exécutions possibles du protocole, les nœuds de ce graphe sont les états globaux du protocole et les arcs sont les différentes transitions (interactions) qui font passé le protocole d'un état global vers un autre.
Un exemple de graphe d'accessibilité est donné sur la figure2.8. Ce graphe correspond au CAEF de la figure 2.7 suivante :

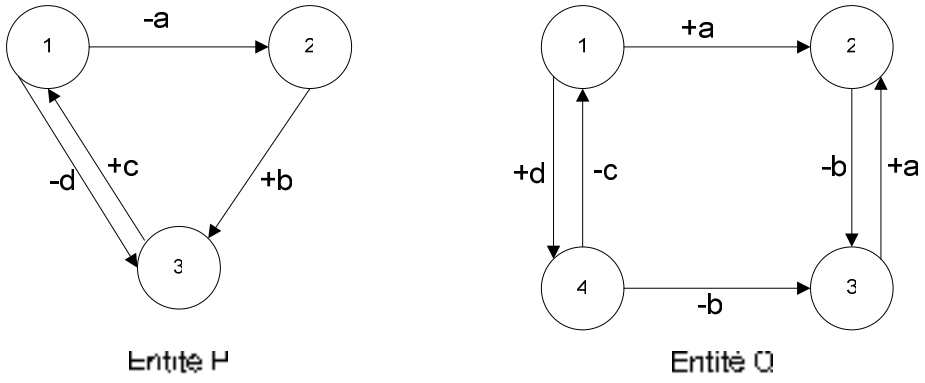


Figure2.7 : Exemple de communication d'automates d'états finis.

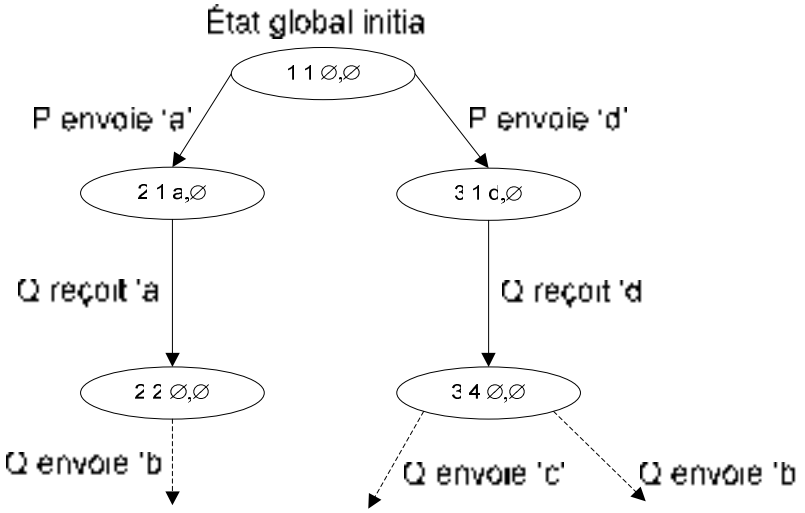


Figure2.8 : Fragment du graphe d'accessibilité correspondant à l'exemple de la fig2.7.

II-7) Conclusion

DynWes définit un modèle de conversation entre services Web à base d'agent, où chaque agent publie la spécification de son service sous forme de CAEF, en suite il suffit pour chaque demandeur de service de télécharger cette spécification et de l'implémenter d'une manière juste à temps pour entrer en interaction avec l'agent fournisseur de service.

Dans un environnement Internet, une spécification publiée par un agent peut contenir des erreurs logiques de protocole. Donc, un agent client doit valider la spécification téléchargée avant de l'implémenter et de rentrer en conversation avec l'agent fournisseur de service.
La validation des protocoles fera l'objet du chapitre suivant où nous donnons un état de l'art des différentes techniques proposées dans ce domaine.

Chapitre III

Etat de l'art sur les techniques de validation de protocoles

Introduction

Une spécification de protocole, doit être validée, pour vérifier qu'elle ne présente pas d'erreurs de conception. Ces erreurs sont détectées par une vérification du graphe d'accessibilité du protocole. Plusieurs techniques ont été proposées pour effectuer cette validation. La première technique qui a été utilisée est l'analyse d'accessibilité standard. Le problème posé par l'application de cette technique est la terminaison de la construction du graphe d'accessibilité. Ce problème est dû à l'explosion combinatoire du nombre d'états globaux générés. Plusieurs techniques ont été proposées pour palier à ce problème en essayant de réduire la taille du graphe d'accessibilité.

Ce chapitre, présente quelques techniques de validation de protocole qui sont connues dans la littérature.

III-2) Techniques de validation de protocole

On peut classer les techniques de validation de protocole, selon leurs méthodes d'exploration de l'espace des états (construction du graphe d'accessibilité), en deux catégories : celles qui utilisent une exploration exhaustive et celles utilisant une exploration partielle du l'espace d'états de protocole.

Nous présentons dans la section suivante quelques techniques basées sur les deux méthodes d'exploration, pour chaque technique nous donnons ses avantages et ses inconvénients.

III-2-1) Techniques de validation exhaustives

Cette catégorie de techniques est utilisable pour des protocoles dont leurs canaux de communication sont de capacité limitée. Leur principe est le parcourt de tous les états possibles du protocole est pour chaque état on effectue un contrôle de validité pour garantir l'absence des erreurs de conception. Bien que cette catégorie de techniques permet la détection de tout type d'erreur logique dans une spécification de protocole, elle est coûteuse en terme de temps d'exécution et en espace mémoire.

Nous avons choisie de présenter trois techniques utilisant l'exploration exhaustive. La première technique est l'analyse d'accessibilité standard qui est la première technique qui a été utilisée pour la validation des protocoles. La deuxième technique est l'analyse structurelle, qui effectue une analyse exhaustive de la conception du protocole est cela après avoir décomposé la spécification globale en plusieurs sous spécifications. La dernière technique est l'analyse avec arbre, qui apporte une amélioration sur la façon de générer le graphe d'accessibilité pour en réduire la complexité de son analyse.

III-2-1-1) Analyse d'accessibilité [18]

L'Analyse d'accessibilité standard [18] était la première approche utilisée pour la validation des protocoles de communication. Les erreurs de conception, telles que les interblocages, les réceptions non spécifiées, les transitions non exécutables et les canaux débordés sont toutes détectables par l'application de cette technique.
Le principe de cette méthode est l'exploration de toutes les interactions possibles entre les entités communicantes du protocole. Pour cela, on génère tous les états globaux accessibles à partir de

l'état global initial du système. A partir de ce dernier toutes les transitions exécutables sont exécutées pour générer de nouveaux états. Chaque nouvel état est généré par l'exécution d'une seule transition à la fois. Cet état est en suite analysé pour vérifier s'il présente une erreur logique de protocole. Ce processus est répété tant qu'il est possible de générer de nouveaux états. A la fin de cette opération on obtient un graphe d'accessibilité de protocole.

La figure3.1 illustre sous forme d'une CAEF une modélisation d'un protocole de communication entre deux entités. L'échange de messages s'effectue à travers des canaux FIFO de capacité un. Le graphe d'accessibilité de ce protocole, construit par une analyse d'accessibilité, est donné sur la figure3.2 dans la page suivante.

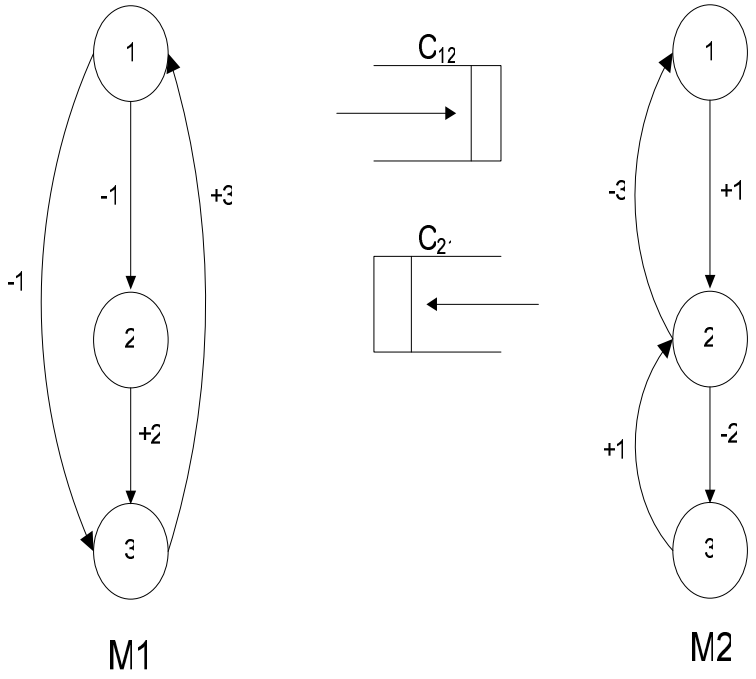


Figure 3.1 : Modélisation d'un protocole sous forme d'AEFC.

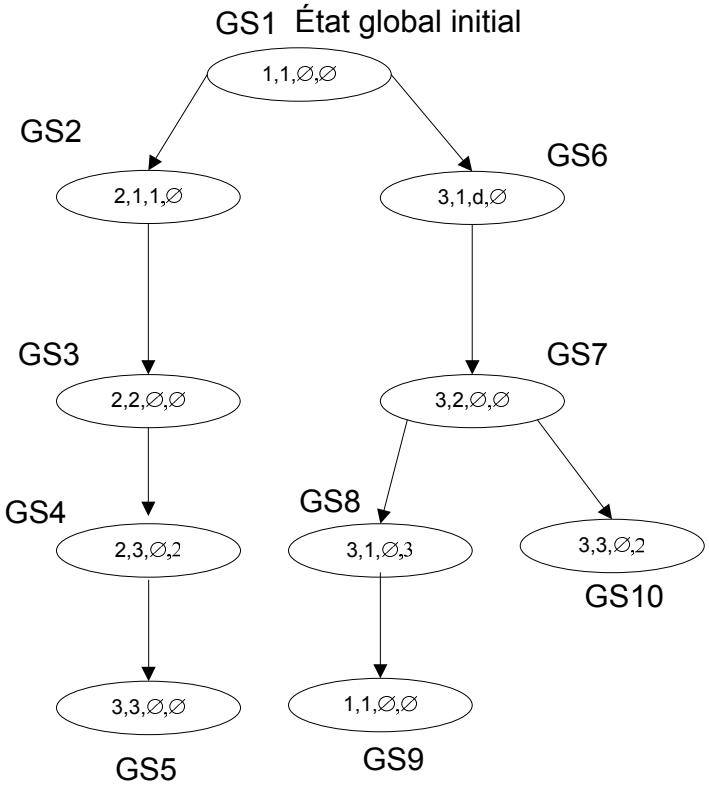


Figure 3.2 : Graphe d'accessibilité du protocole de la figure3.1.

Un cas d'erreur est détecté dans l'état globale GS5 sur le graphe d'accessibilité ci-dessus. Dans cet état global, toutes les entités sont en attente de réception d'un message avec leurs canaux de communication vides, donc une erreur d'interblocage est détectée.

Bien que cette méthode est facilement automatisable et permet la détection de toutes les erreurs logiques de protocole, elle soufre du problème de l'explosion combinatoire dans le cas d'analyse des protocoles complexes. Ce problème est dû au grand nombre d'état global à générer pendant l'analyse.

III-2-1-2) Analyse structurelle

L'analyse structurelle [5,6] permet de réduire la complexité de l'analyse d'un protocole par une division du système à analyser en plusieurs sous système. Chaque sous système et ensuite validé par une analyse d'accessibilité indépendamment des autres sous système. Une fois l'analyse des sous système est terminée les différents résultats sont combinés afin de valider le protocole en entier.

Cette technique est applicable pour des protocoles équitables ; où chacun des AEF communicant est isomorphisme (a une relation de composition un à un). Cette technique utilise une approche de décomposition de graphe pour décomposer le graphe du protocole à analyser en sous graphes. Chaque sous graphe doit avoir un nœud d'entrée et un ou plusieurs nouds de sortie. Dans le cas où un sous graphe ne contient pas de nœuds de sorties son nœud d'entrée est considéré comme étant aussi un nœud de sortie. Vers la fin de l'analyse, Les sous graphes générés sont connectés entre eux pour former le graphe d'accessibilité du protocole. Un sous graphe est connecté à un autre par la liaison du nœud de sortie du premier sous graphe à un nœud d'entrée du deuxième sous graphe.

III-2-1-3) N-tree validation [1]

La technique de validation N-tree[1], considère l'exécution du chaque entité de protocole séparément, au lieu de considérer l'exécution des différentes entités ensemble.
Pour chaque entité, on lui génère son arbre d'exécution. Une fois cette opération est achevée, on prend les arbres d'exécution obtenus, qui sont appelé N-tree, pour construire ce qu'on appel un arbre de protocole (tree-Protocol).
Cette technique redirige le problème de la détection des erreurs de conception, vers l'identification de toutes les transitions de réception exécutables et tous les états globaux stables du protocole ; un état global stable est constitué des états accessibles des entités du protocole avec leurs canaux de communication vides.

Pour n'importe quel protocole, on peut construire son Tree-Protocol en traçant toutes ses transitions exécutables. Inversement, disposant d'un Tree-Protocol on peut facilement construire le protocole qui lui correspond par une opération d'éclatement des états et des transitions équivalentes qui le compose.
Dans la construction d'un Tree-Protocol , à chaque fois qu'un même état ou transition sont répétés dans l'arbre, on les renomme de façon à ne pas avoir des branches dans l'arbre dont le nom de leurs états ou transitions se répètent. Par exemple, si on a un état S d'une entité d'un protocole, à chaque fois que cet état est utilisé dans l'arbre (accessible depuis plusieurs chemins) on lui affecte un nom différent. Par exemple S.0 pour la première apparition, S.1 pour la deuxième …etc. On fait la même chose pour les transitions qui apparaissent plusieurs fois. Pour une transition d'envoi de message –m, et lors de la construction de tree-protocole, on étiquette sa première apparition par –m.0 , pour une deuxième apparition -m.1 … etc.
Sur la figure3.4 et 3.5 respectivement, sont donnés les N-tree obtenus par l'application de cette technique sur la spécification des entités 1 et 2 du protocole de la figure3.3.
La génération des branches dans la construction d'un arbre d'une entité communicante dans un protocole, se termine lorsque un nouveau nœud est créé et son analyse indique qu'il est déjà

répété avec les mêmes transitions. Cela veut dire que la branche qui va être développer à partir de ce nœud a été déjà construite auparavant, et donc on a pas besoin de la développer encore une autre fois.

Les auteurs de cette technique ont défini trois conditions pour qu'une transition soit exécutable : la première condition dit qu'une entité ne peut pas recevoir un message depuis une autre entité que après avoir déjà reçu tous les messages précédemment envoyés par cette entité. La deuxième condition suppose qu'un protocole est construit de plus de deux entités. Si une entité1 attend à recevoir un message *m* depuis une autre entité2 à l'état X, et que l'entité2 passe à l'état Y après avoir envoyer ce message *m*, alors les deux derniers états qu'une troisième entité 3, doit avoir accédé, avant que l'entité 1 accède à son état X et l'entité 2 accède à son état Y doivent être en relation de prédécesseur-successeur. La troisième condition indique que pour qu'une entité 1 peut recevoir un message depuis une autre entité 2 à l'état X ; le dernier état que l'entité 1 doit accéder avant que l'entité 2n'envoie le message *m* doit être le prédécesseur de l'état X. autrement dit, l'entité 1 doit avoir déjà passé l'état X au moment de l'envoi du message *m*. donc il est impossible pour l'entité 1 de recevoir le message *m* à l'état X.

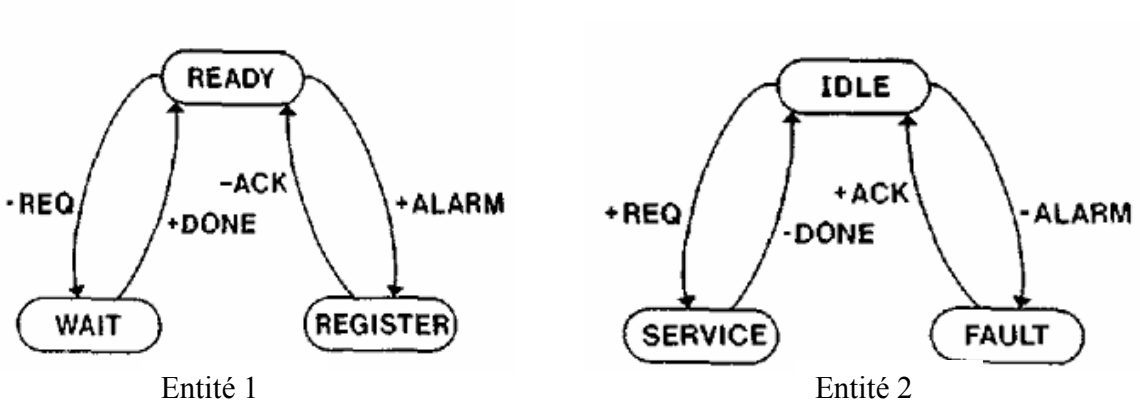


Figure 3.3: Spécification d'un protocole de communication entre deux entités.

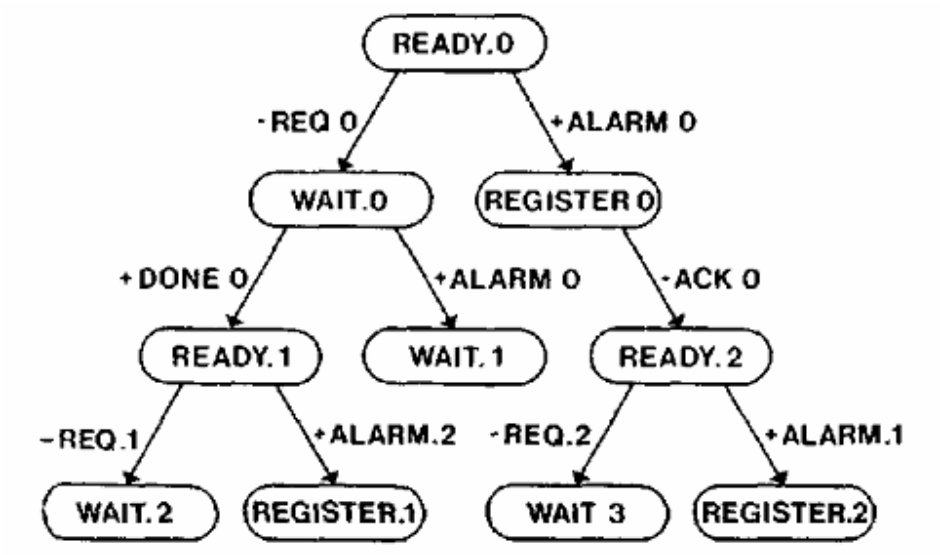


Figure 3.4: Tree protocole de l'Entité 1

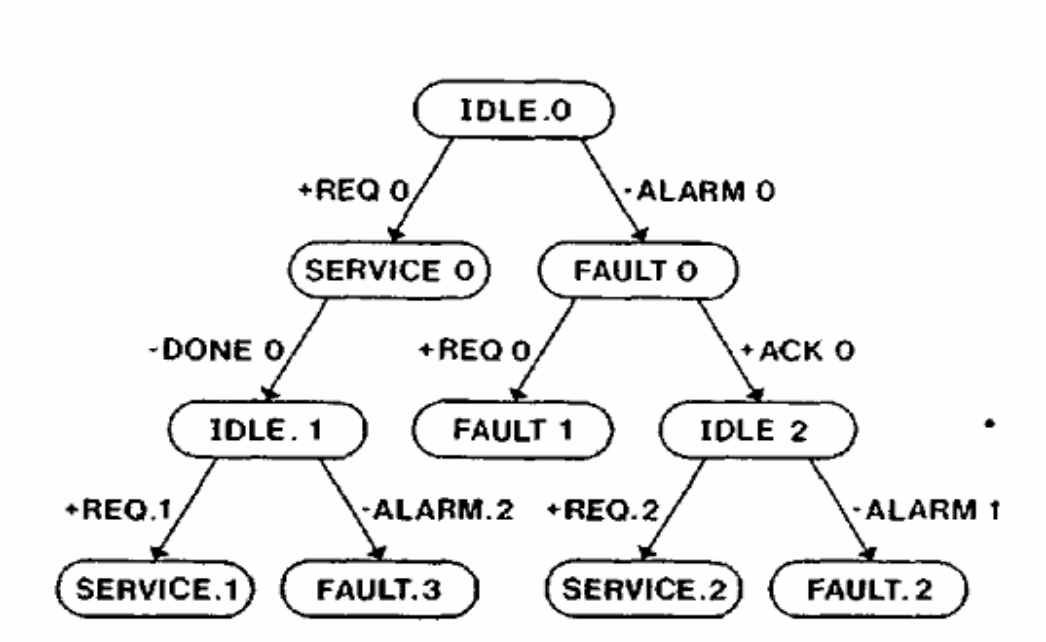


Figure 3.5 : Tree protocole de l'Entité 2

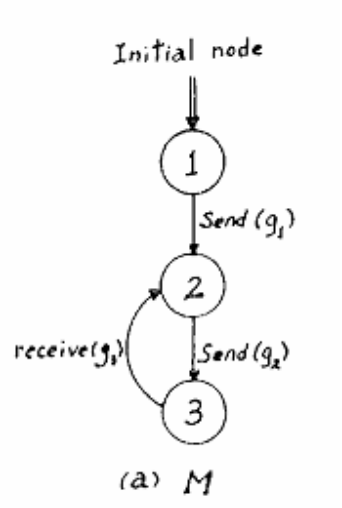
Cette technique présente l'avantage de diviser le problème de la validation d'un protocole en sous problèmes, ce qui facilite l'opération de validation. En revanche elle présente une limitation concernant le développement de l'arbre qui est indécidable.

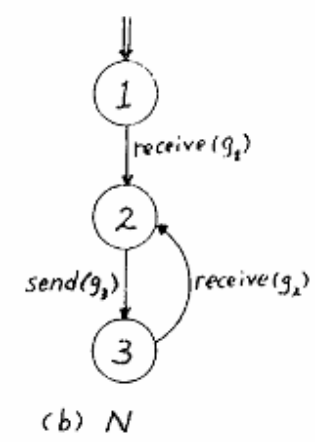
III-2-2) Techniques utilisant une exploration partielle de l'arbre de protocole

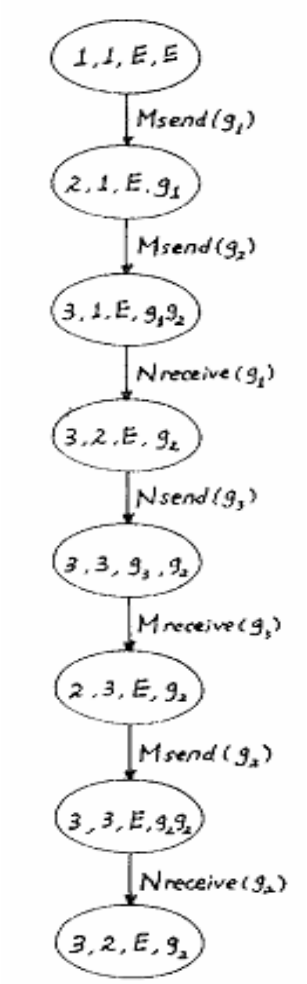
Nous présentons dans cette section quelques techniques de validation de protocole qui se basent sur des approches qui effectuent une exploration partielle du graphe d'accessibilité tout en essayant de détecter le maximum d'erreurs logiques.

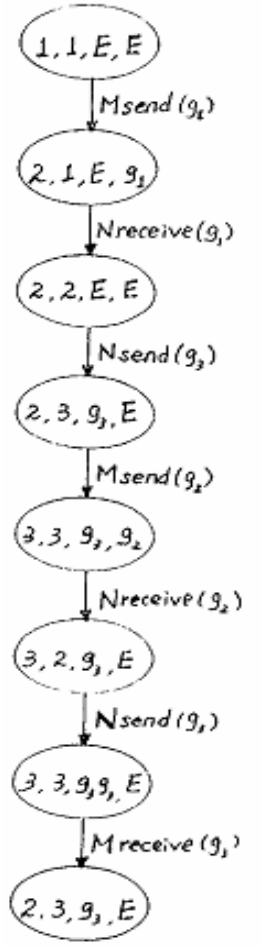
III-2-2-1) Maximal Progress State Exploration [8]

Cette technique a été proposée pour valider des protocoles de communication entre deux entités seulement. Avec cette technique, le processus de validation est divisé en deux sous taches qui doivent s'exécuter en parallèle. Les auteurs de cette technique ont supposé que le majeur problème de la validation exhaustive c'est quelle suppose que pour générer l'espace des états de protocole, on doit au premier lieu considérer la progression des entités communicantes dans leurs différents ordres possibles. Ce qui conduit à la génération de plusieurs états inutiles pour l'analyse car un même état peut être généré par différentes progressions durant l'analyse du protocole.
Les erreurs traitées par cette technique sont de type interblocage, réception non spécifiée et canal débordé. Pour chaque entité, on génère et on examine les états globaux qui résultent de sa progression. Un état qui ne présente pas d'erreur est dit état progressant sinon il est dit non progressant.
Soit un protocole P modélisé par une communication de deux AEF, pour chaque entité on crée une sous tache pour la génération des états globaux résultants de la progression de cette entité. La première sous taches T1 force l'exécution de l'entité 1 jusqu'à ne plus pouvoir progresser (doit recevoir un message pour pouvoir continuer son exécution). Dans ce cas la, on fait exécuter l'entité 2 pour recevoir le message nécessaire pour que l'entité 1 puisse continuer sa progression. Le processus est répété jusqu'à la fin de l'exécution de l'entité1. Le même processus est appliqué pour la génération des états de la deuxième entité par la sous tache T2.
La composition des deux graphes obtenus par l'application des deux opérations précédentes, donne le graphe d'accessibilité du protocole.
Sur la figure3.6[8] nous donnons une modélisation d'un protocole de communication entre deux entités M (figure 3.6.a) et N (figure 3.6.a) . Le résultat de L'application des deux sous taches de la méthode Maximal Progress est donné sur les figures 3.6.c et 3.6.d pour les deux entités M et N respectivement.Le graphe d'accessibilité du protocole est enfin obtenu par la composition du résultat d'exécution de chaque sous tache.









(c) Génération des états de l'entité M (d) Génération des états de l'entité N

Figure3.6 : exemple de validation maximal progress.

III-2-2-2)PROVAT: PROtocol Validation Testing [18]

PROVAT est une technique basée sur des méthodes de recherche utilisées dans le domaine de l'Intelligence Artificielle pour la recherche des erreurs de conception. Elle effectue une exploration partielle du graphe d'accessibilité, et on se basant sur des heuristiques, elle essaye de parcourir les branches qui peuvent présenter des erreurs logiques de protocole.

Dans cette technique, le choix de la branche à explorer est décidé par rapport à des heuristiques. Il existe trois points où une heuristique peut être appliquée :

1-Decider du prochain noeud de l'arbre à développer.

2-Dans le développement d'un nœud, quel est le successeur à parcourir (au lieu de parcourir tous les successeurs).

3-Quelles sont les branches à élaguer de l'arbre.

Cette technique est utile dans des environnements où on est contraint par le temps/espace mémoire. Son inconvénient majeur est qu'on doit effectuer une recherche à part pour chaque type d'erreur, ce qui rend l'opération de recherche compliquer.

III-2-2-3) Analyse d'accessibilité réversible (Reverse Reachability Analysis) [14]

L'analyse d'accessibilité réversible (RRA[6])[14] génère l'espace des états de protocole d'une manière réversible. A partir d'un ensemble d'états, dit indésirable, on essaye d'accéder l'état global initial par une opération de retour arrière. L'existence d'un chemin de retour de l'état indésirable vers l'état initial, confirme cet état indésirable comme étant une erreur d'interblocage, si non l'état est dit ne présente pas d'erreur. L'ensemble des états indésirables est composé des états globaux qui présentent des caractéristiques d'un état d'interblocage (chaque entité est dans un état d'attente de réception d'un message et l'état de leurs canaux d'entrée est vide).

Pour le protocole de la figure3.7, la RRA génère l'état globale <2,2,0,0> comme étant un état indésirable. Lorsque on génère le graphe d'accessibilité réversible on obtient le chemin de retour suivant : $<2,2,0,0> \rightarrow^{R} <1,2,0,c> \rightarrow^{R} <1,1,a,c> \rightarrow^{R} <0,1,0,c> \rightarrow^{R} <0,0,0,0>$. Donc l'état global initial est accessible depuis l'état indésirable <2,2,0,0>, ce qui confirme la présence d'une erreur d'interblocage dans cet état global.

[6] Reverse Reachability Analysis

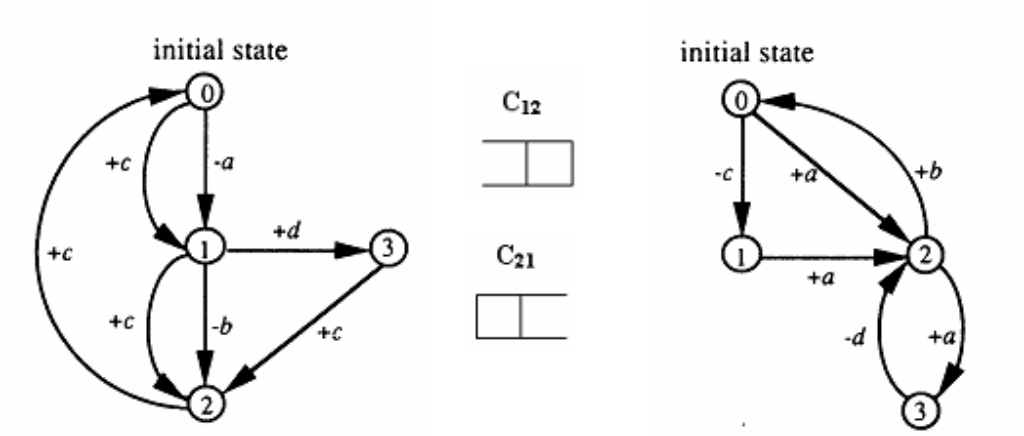


Figure 3.7 : une spécification de protocole[14].

Cette technique parcourt un nombre d'états plus petit que celui parcouru par une analyse exhaustive, ce qui permet de gagner en complexité temps d'exécution et espace mémoire. Mais son application est limitée pour la détection des erreurs de type interblocage.

III-2-2-4) Exploration aléatoire des états de protocole (RWSE[7]**)** [36]

Cette technique peut être vue comme étant une analyse d'accessibilité réduite ; au lieu de parcourir toutes les branches du graphe de protocole, on se contente de parcourir seulement quelques branches choisies d'une manière aléatoire. Les auteurs de cette technique, suppose qu'une analyse partielle du fonctionnement de protocole suffit pour détecter un nombre important d'erreur, cela permet de réduire la complexité de calcul tout en assurant un bon fonctionnement du protocole.
Bien que cette technique s'avère très simple, elle ne spécifie pas des conditions de terminaisons. Dans l'analyse d'accessibilité exhaustive, le processus de vérification s'arrête une fois tout le graphe est parcouru. Dans l'analyse RWSE, puisque on fait un parcourt aléatoire, il est impossible de décider que l'analyse va se terminer après un parcourt de telle ou telle branche ou bien de dire que toutes les erreurs sont détectées. Donc, cette technique permet une détection des erreurs de conception mais ne garantie pas que le protocole ne va pas présenter d'autres erreurs lors de son exécution.

[7] Random Walk State Exploration

III-2-2-5)Analyse d'accessibilité simultané (Simultaneous rechability analysis) [24]

SRA[8] [24] est une technique proposée pour la détection des erreurs d'interblocage. Dans cette technique, on génère et on analyse un sous ensemble de l'ensemble des états globaux du protocole. A partir de l'état global initial et avant chaque génération d'un nouvel état, on effectue un calcul des transitions qui peuvent être exécutées simultanément à partir de cet état. Ces transitions sont exécutées simultanément pour générer un seul état global au lieu de plusieurs états globaux pour chaque exécution d'une transition.

Dans cette technique, on identifie trois types de transition : les transitions indépendantes, les transitions dépendantes et les transitions de réception impossible.
On dit qu'une transition est indépendante dans un état global, si elle peut s'exécuter immédiatement sans avoir à attendre l'exécution d'une autre transition. Une transition d'envoi de message est dite indépendante, si l'envoi du message correspondant à cette transition ne va pas déborder son canal de sorti. Une transition de réception de message est dite indépendante, si son canal d'entrée contient le message attendu par cette transition.
Les transitions dépendantes sont des transitions qui ne peuvent pas s'exécuter immédiatement car elles dépendent de l'exécution d'autres transitions. Une transition d'envoi de messages est dite dépendante si son canal de sortie est plein est donc la transition ne peut pas envoyer son message sur ce canal. Une transition de réception de message est dite dépendante lorsque le contenu de son canal d'entrée est vide, si le message en tête de son canal d'entrée est différent du message attendu, on dit que cette réception est impossible.

A partir d'un état global, l'union des transitions indépendantes et des transitions dépendantes forme un ensemble candidat, si il inclus : au plus une transition dépendante ou indépendante de chaque entité du protocole et au moins une transition indépendante. La suppression des transitions dépendantes dans un ensemble candidat, donne un ensemble de transitions indépendantes.
Cette technique permet de réduire l'espace des états globaux en générant et en analysant, seulement, les états globaux obtenus par l'exécution d'un ensemble de transitions indépendantes. Donc, le passage d'un état global vers un autre s'obtient par l'exécution des transitions simultanées calculées à partir de cet état global.
La figure3.10 illustre l'application de cette technique sur le protocole du communication de la figure3.8. Dans cet exemple, un état global est représenté sous forme de deux vecteurs, le premier désigne les entités du protocole et le deuxième vecteur représente les différents canaux de communication. L'état global initial de l'exemple est représenté par :

[8] Simultaneouse rechability analysis

(<P_1=1,P_2=4,P_3=7>,<C_{12}=ε,C_{13}=ε,C_{21}=ε,C_{23}=ε,C_{31}=a,C_{32}=c>). L'ensemble des transitions de ce protocole est composé de t_1=(1,+a_{31},2), t_2=(4,+c_{32},5), t_3=(4,-d_{23},6) et t_4=(7,-b_{31},8) et t5=(1,+b_{31},3). Les transitions t_1,t_2,t_3 et t_4 sont des transitions indépendantes, t_5 est une transition impossible (car l'entête du canal C_{31} contient le message 'a' initialement).
A partir de ces transitions, on construit deux ensembles de transitions simultanément exécutables, qui sont : {t_1,t_2,t_4} et {t_1,t_3,t_4}. Ces deux ensembles vont servir dans la construction du graphe SRA. Les transitions de chaque ensemble sont exécutées simultanément pour générer un seul nouvel état global pour chaque ensemble. De cette manière, on évite de générer un état global pour chaque exécution d'une transition à la fois. Le graphe obtenu est donné sur la Figure3.9.

Si on compare le résultat obtenu par rapport à la méthode d'analyse d'accessibilité (donnée sur la figure3.10), on voie clairement que la technique SRA permet de réduit le nombre d'états à vérifier de 20 à 3 états seulement, ce qui représente un gain important en complexité d'analyse.

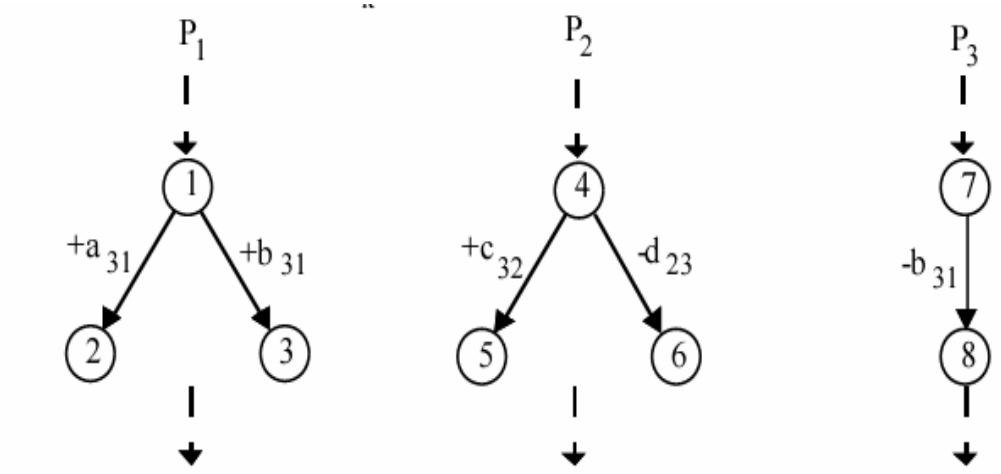


Figure 3.8 : un fragment de modélisation de protocole sous forme d'AEF

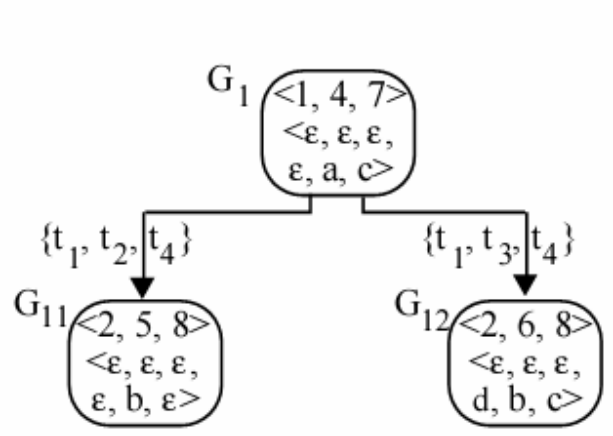


Figure 3.9 : résultat de l'application de la technique SRA

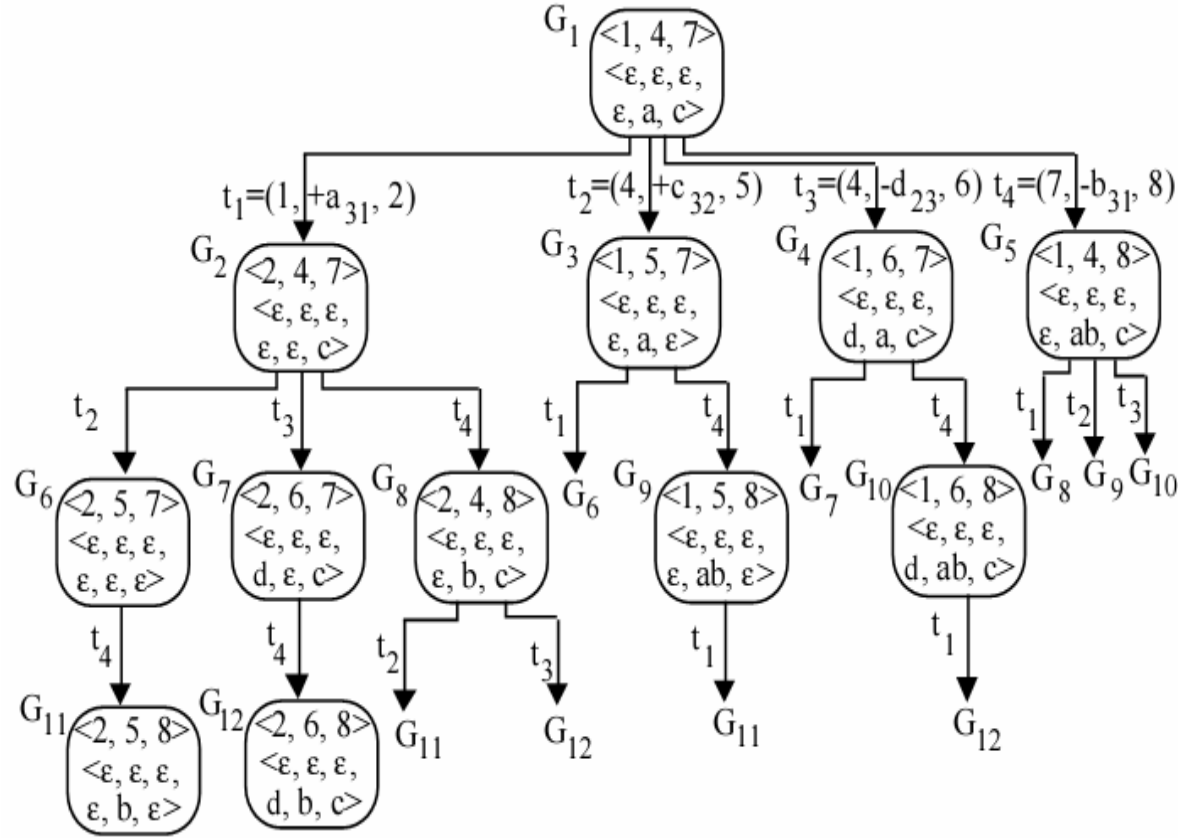


Figure 3.10 : résultat de l'application de l'analyse d'accessibilité.

L'avantage que présente la technique SRA c'est qu'elle évite de générer les états globaux inutiles pour l'analyse. Avec cette approche, l'espace des états du protocole sera réduit considérablement. Plus le protocole à un comportement concurrentiel plus cette technique profite de cette concurrence pour réduire le nombre d'état global à analyser. Pour un protocole qui n'implémente pas de concurrence, le coût de cette technique sera égal au coût d'une analyse d'accessibilité exhaustive.

III-2-2-6)Enhancing Compositional Reachability Analysis with Context Constraints[4]

Cheng et ***Kramer***[4], proposent une amélioration sur les technique d'Analyse d'Accessibilité Compositionnelle (AAC)[25]. L'AAC est une technique utilisée dans la modélisation des systèmes concurrents pour introduire de la modularité et contrôler le problème de l'explosion combinatoire. Son principe est la décomposition du système à modéliser en une hiérarchie de sous systèmes minimisés en utilisant des techniques algébriques. Pour avoir le système complet, on procède à une opération de composition des différents sous systèmes. Bien que cette méthode apporte de la modularité sur la spécification des protocoles, le problème de l'explosion combinatoire reste toujours présent comme les sous systèmes peuvent contenir des états et transitions qui ne sont pas tolérés par leur environnement ou leur contexte global de fonctionnement.

L'amélioration apportée par ***Cheng*** et ***Kramer*** est de prendre en compte, lors de la construction des sous systèmes, les contraintes contextuelles. Ces contraintes sont introduites sous forme d'interfaces de processus (système de transition étiqueté) qui va être combiné avec l'automate du sous système à analyser. Cette combinaison permet de réduire le nombre d'état et de transition dans un premier niveau dans la construction du graphe d'accessibilité de système complet, est donc réduire la complexité de l'analyse AAC.

III-2-2-7)Analyse d'accessibilité équitable (Fair Reachability analysis)

Rubin et West[15], proposent une nouvelle technique d'exploration des états de protocole. Leur technique se base sur la notion de séquence canonique. Une forme canonique ,d'une séquence d'étape d'exécution entre deux entités d'un protocole, peut être obtenue par le choix d'une paire d'exécution du protocole tel que chaque paire d'exécution contient une étape d'exécution d'une entité du protocole. Dans chaque paire d'exécution, on choisie d'abord une première étape d'exécution du premier processus, à moins que cette étape est un événement de réception de message depuis un canal vide. Si c'est le cas, l'étape d'exécution du deuxième processus est choisie d'abord. Deux séquences d'exécution sont canoniquement équivalentes si elles ont la

même forme canonique. Il a était montré [15] que l'équivalent canonique de n'importe quelle séquence exécutable est une séquence exécutable. Pour introduire le concept de séquence canonique dans l'algorithme de l'exploration de l'espace des états, il est nécessaire de diviser l'exécution du protocole en une série d'interactions basiques. Une interaction est une séquence d'étapes d'exécution qui fait changer l'état du protocole d'un état stable vers un autre état stable. Un état stable est défini par l'état que prennent les entités du protocole lorsque leurs canaux de communications sont vides. Une interaction basique est une interaction qui ne peut pas être divisée en simples interactions sans changer la séquence d'étapes d'exécution des deux entités du protocole. L'algorithme proposé détermine une interaction basique par une opération de couplage des paires de transitions des deux entités, exécute chaque interaction basique dans sa forme canonique et prend en considération les états intermédiaires (avec des canaux non vides) lorsque c'est nécessaire. En exécutant seulement des séquences canoniques, cela permet de réduire significativement le nombre d'état global à analyser, puisque on ne génère que les états qui ont un nombre de messages égale sur les deux directions.

Ce travail a été ensuite amélioré par Gouda et Han[20] pour déterminer laquelle des deux entités du protocole est de capacité limitée. Ces auteurs ont observé que l'exécution d'une séquence canonique force les deux entités à progresser avec une même vitesse, ils ont donné le non de **Analyse d'accessibilité équitable** (notée FRA[9]) pour la technique proposée dans [15]. Le graphe d'accessibilité construit par la technique FRA est appelé graphe d'accessibilité equitable (noté FRG[10]). Pour déterminer si les canaux de communication entre les deux entités sont bornés, le FRG est construit ensuite augmenté, de telle façon que le graphe augmenté peut être utilisé pour déterminer l'accessibilité des nœuds du protocole.

Dans la référence [10], Liu et miler propose une généralisation du FRA sur des protocoles de communication cycliques entre n entités. Les entités d'un protocole cyclique sont connectées entre elles via un anneau unidirectionnel. Ils ont prouvé que pour un protocole cyclique et lorsque l'espace de ses états accessibles par l'application du FRA est fini, la détection des interblocages est décidable avec le FRG et la détection des autres types d'erreurs logiques, telles que les réceptions non spécifiées et les transitions non exécutables, est aussi décidable via une extension du FRG.

[9] Fair Reachability Analysis
[10] Fair Reachability Graphe

III-2-2-8) Approche de validation de protocole basée sur l'exploration des Automates complexes (CPVA-ECSM[11]) [30]

Cet article propose une technique de validation des protocoles modélisés sous forme d'une communication d'automates d'états complexes. La technique utilise une exploration partielle de l'espace des états pour la détection des états d'interblocage. Le type d'erreur d'interblocage est aussi détecté, qu'il s'agit d'un interblocage simple, hybride ou complexe.
Cette technique agit en deux phases : la première phase consiste en la détection des états susceptibles de provoquer un interblocage, en suite par une opération de retour arrière, la deuxième phase est entamée pour confirmer les états qui présente un vrai interblocage.
La complexité de cette méthode est proportionnelle au nombre d'états susceptibles de provoquer un interblocage et qui est dans la plupart des cas inférieur au nombre totale d'états composant le système.
Cette technique présente des limitations du fait qu'elle n'est pas applicable que pour des modélisations entre deux entités seulement. En plus qu'elle ne supporte que des canaux de communication de capacité un.

III-3) Conclusion

Dans ce chapitre, nous avons présenté certaines techniques de validation de protocoles existantes dans la littérature. Pour cela, nous les avons classées, selon leurs méthodes de l'exploration du graphe d'accessibilité, en deux catégories : celles utilisant une exploration exhaustive [1,5,6,8,18] et celles utilisant une exploration partielle[14,18,24,36,30,4,10,15,20]. Les techniques utilisant une exploration partielle essayent d'optimiser la construction du graphe d'accessibilité tout en analysant tout l'espace des états globaux du protocole. Par conséquent, le problème de l'explosion combinatoire reste toujours présent, ce qui limite leur applicabilité à des protocoles simples. Les techniques utilisant une exploration partielle, réussissent à réduire la taille du graphe d'accessibilité mais cela n'est obtenu qu'avec un contre partie qui est : soit de limiter les types d'erreurs traités ou bien se contenté de valider une partie du protocole. L'objectif principal est de palier au problème de l'explosion combinatoire posé par l'utilisation de l'analyse standard. Toutes les améliorations qui ont été apportées avaient comme objectif de réduire la taille du graphe d'accessibilité tout en gardant une détection d'un maximum d'erreurs logiques. Pour chaque technique exposée, nous avons expliqué son principe, ces avantages et ses inconvénients.

Dans le chapitre suivant, nous proposons une nouvelle technique de validation de protocoles qui profite des avantages des techniques de validation avec retour arrière et des techniques de construction de graphe de sauts afin d'apporter une réduction importante sur la taille du graphe d'accessibilité à explorer.

[11] A Communication Protocol Validation Approach based on Exploration of Complex State Machines.

Chapitre IV

Proposition d'une technique de validation de Protocoles

IV-1) Introduction

Dans ce chapitre, nous présentons notre technique de validation de protocoles que nous avons appelée : **Reverse Leaping Reachability Analysis** (RLRA) ; Analyse d'accessibilité basée sur des sauts réversibles.
Notre méthode de validation fait partie de la famille des techniques avec retour arrière [14,16]. Le principe de retour arrière est le parcours inverse des chemins d'un graphe d'accessibilité.
À partir d'un état suspect, on cherche à atteindre l'état initial du protocole (racine du graphe). Seuls les états, pour les quels un chemin vers la racine est trouvé, sont confirmés erreur de protocole. Notre technique est basée sur le même principe de retour arrière, mais au lieu de travailler sur un graphe d'accessibilité standard (couvre tout l'espace des états), nous utilisons un graphe de sauts (LRA[12])[17]. Ce graphe est construit à partir d'un sous ensemble de l'espace des états du protocole. Les états d'interblocage, s'ils existent, appartiennent toujours à ce sous espace.
L'utilisation d'un graphe de sauts réduit le nombre d'état à parcourir tout en gardant la capacité de détecter toutes les erreurs d'interblocage, ce qui permet de réduire la complexité de l'analyse du protocole.

Dans le modèle de CAEF que nous utiliserons, les traitements internes aux agents, ne seront pas modélisés dans la spécification ; seuls les traitements qui nécessitent une interaction entre agents seront modélisés.

Avant de détailler notre proposition, nous expliquons dans la première section la technique de construction des graphes de sauts. Dans la deuxième section, nous expliquons la façon de construire des chemins inverses d'un graphe de sauts. La capacité de notre technique à détecter les erreurs d'interblocage est prouvée à travers le théorème principal de notre proposition. Nous terminons ce chapitre par l'algorithme détaillé de notre technique.

[12] LRA :Leap Reachability Analysis.

IV-2) Notation

Dans le reste de ce mémoire nous utilisons les notations et définitions suivantes :

Soit I={1,2,…,n} un ensemble fini d'indices, avec cardinal(I)≥2. Dans un modèle de Communication d'automate d'états finis, un protocole $\prod$ est une paire (P,L), où P={P_i / i∈I} est l'ensemble des n entités communicantes du protocole et L⊆IxI est la relation d'incidence non réflexive qui identifie un ensemble non vide de canaux de communication {C_{ij} /(i,j)∈L}. Chaque canal C_{ij} ((i,j)∈L) est une file FIFO, bornée et sans erreurs reliant l'entité P_i à l'entité P_j.

Chaque entité P_i est un automate d'états finis: (S_i,$S0_i$,E_{fi},M_i,Δ_i) (simple ou complexe-aplati) composé d'un ensemble d'états S_i , d'un ensemble d'états finaux E_{fi} ,d'un alphabet de messages M_i ,d'un ensemble de transitions Δ_i et d'un état initial $S0_i$.
M_i est l'ensemble de tous les messages que peut échanger une entité P_i (alphabet de l'automate)
Le contenu du canal C_{ij}, noté c_{ij}, est une suite de messages de M_i, i.e. $c_{ij}\in M_i^*$. Chaque canal peut contenir un nombre maximum de K (entier positif) messages. Une entité P_i envoie des messages sur son canal de sortie $C_{ij}(j\in I\wedge j\neq i)$ et reçoit des messages sur ses canaux d'entrées $C_{ji}(j\in I\wedge j\neq i)$.
La relation de transition Δ_i est définie par :$\Delta_i : S_i x M_i \rightarrow Si$. Une transition $t=(s_i,\mu,s_i')\in \Delta_i$, définie à l'état local s_i de l'entité P_i, est dite transition d'envoi de message lorsque μ=-x, indiquant un envoi d'un message x (x∈M_i) par l'entité P_i sur le canal C_{ij}. *t* est dite transition de réception de message lorsque μ=+y, indiquant une réception d'un message y (y∈M_i) par l'entité P_i depuis le canal C_{ji}. Les signes – et + indiquent respectivement un envoi et une réception de message.

Un état global G du protocole $\prod$ est représenté par une paire ($\langle s_i^G\rangle_{i\in I}$,$\langle c_{ij}^G\rangle_{(i,j)\in L}$), où s_i^G est l'état local de l'entité P_i et c_{ij}^G est le contenu du canal C_{ij} à l'état global G.
L'état global initial est notée $G^0=(\langle s_i^{G^\circ}\rangle_{i\in I},\langle c_{ij}^{G^\circ}\rangle_{(i,j)\in L})$, tel que $s_i^{G^\circ}= S0_i$ et $c_{ij}^{G^\circ}=\varepsilon$, pour tout i∈I et (i,j)∈L.

Nous définissons la fonction *act()* comme suite [17] :
∀ i,j∈I, $\Delta_i\cap\Delta_j=\varnothing$;
1- Pour une transition t∈Δ_i , act(t)= i est l'indice de l'entité auquel appartient la transition t,

2- Pour un ensemble de transition T, act(T)={i∈I /T∩$\Delta_i\neq\varepsilon$} est l'ensemble des indices des entités qui ont au moins une transition dans T ;

Nous définissons également la fonction Front(c_{ij}^G) qui renvoie le message en tête du canal c_{ij} à l'état global G.

IV-3) Graphe de sauts [17]

Un graphe de sauts est un graphe d'accessibilité réduit. Cette réduction est obtenue du fait que le passage d'un état global vers un autre peut se faire via l'exécution de plusieurs transitions à la fois, ce qui permet d'éviter de générer des états intermédiaires résultants de l'exécution de chaque transition à part. les transitions qui peuvent s'exécuter en même temps sont appelées : transitions simultanément exécutables. Ces transitions simultanées forme un saut dans un graphe d'accessibilité.

IV-3-1) Transitions simultanément exécutables

Pour calculer les transitions simultanément exécutables dans la construction d'un graphe de sauts, nous avons besoin d'abord de définir les notions de transitions exécutables et transitions potentiellement exécutables.

Transitions exécutables

Une transition $t=(s_i,\mu,s_i')$ d'une entité communicante P_i ,est dite exécutable à l'état global G, ssi $s_i=s_i^G$ (i.e s_i est l'état local de l'entité P_i à l'état global G) et *t* vérifie une des deux conditions suivantes :

1- *t* est une transition d'envoi de message et le contenu du canal C_{ij} est inférieure à K, i.e. $|C_{ij}|<K$. ou,

2- *t* est une transition de réception de message tel que $\mu=+y$ avec $y\in M_i$ et $Front(c_{ji}^G)=y$. Autrement dit, *t* peut recevoir le message qui est en tête du canal c_{ji}.

L'ensemble des transitions d'envoi ou de réception de messages de l'entité P_i, qui sont exécutables à l'état globale G, sont notés respectivement $X_i^-(G)$ et $X_i^+(G)$. L'union de ces deux ensembles est noté $X_i(G)= X_i^-(G)\cup X_i^+(G)$. L'ensemble de toutes les transitions des différentes entité communicantes du protocole, qui sont exécutables à l'état G, est noté : $X(G)= \cup_{i\in I}X_i(G)$.

Transitions potentiellement exécutables

Une transition de réception de message $t=(s_i,+y,s_i')\in \Delta_i$ est dite potentiellement exécutable à l'état global G, ssi $s_i=s_i^G$ et $c_{ji}^G=\varepsilon$. Une transition d'envoi de message $t=(s_i,-x,s_i')\in \Delta_i$, avec $x\in M_i$, est dite potentiellement exécutable à l'état global G, ssi $s_i=s_i^G$ et $|c_{ij}|=K$; K est la capacité des canaux de communication du système.

L'ensemble des transitions de réception de message et des transitions d'envoi de message de l'entité P_i qui sont potentiellement exécutables à l'état globale G, sont noté respectivement $P_i^+(G)$ et $P_i^-(G)$. L'ensemble de toutes les transitions potentiellement exécutables de l'entité P_i est noté : $P_i(G)= P_i^+(G)\cup P_i^-(G)$.

Lors de la construction d'un graphe de sauts, les transitions à exécuter simultanément, doivent être exécutables dans le même état global et doivent appartenir à des entités du protocole qui ne disposent pas de transitions potentiellement exécutables[17]. Un ensemble de telles transitions est appelé saut, noté *pleap* (proper leap).

Donc, un saut est un ensemble non vide de transitions : $T\subseteq X(G)$, tel que pour toute transition $t,t'\in T$, $act(t)\neq act(t')$ si $t\neq t'$ (i.e T contient au plus une transition exécutable de chaque entité). Les différents ensembles T, construits à l'état global G, forment un ensemble de sauts noté *pleap(G)*.

Définition1 : Calcul de l'ensemble pleap (sauts)[11]
Soit G un état global.
On note par *Wait*(G), l'ensemble des indices des entités qui disposent d'une transition potentiellement exécutable à l'état G. Cet ensemble est définit par :
$Wait(G)=\{i \in I / X_i(G) \neq \emptyset \Rightarrow P_i(G) \neq \emptyset\}$.
L'ensemble des sauts *pleap*(G) est donc calculé comme suite :

$$pleap(G)=\begin{cases} \{ \prod_{i \notin Wait(G)} X_i(G) \} \; Si \; wait(G) \subset I \\ \{\{t\} / t \in X(G)\} \; SiNon \end{cases}$$

Avant de calculer l'ensemble *pleap*(G) , on doit calculer l'ensemble *Wait* pour identifier les entités qui disposent de transitions potentiellement exécutables. Si toutes les entités appartiennent à l'ensemble *Wait* ça veut dire qu'il n'y a pas de saut à faire à partir de G et dans ce cas la on exécute chaque transition de X(G) indépendamment des autres (transitions simples). Si l'ensemble *Wait* est inclut dans I, l'ensemble de sauts à exécuter à partir de G (pleap(G)) sera le produit cartésien des ensembles de transitions exécutables de chaque entité (X_i(G)) i.e $pleap(G)=\prod_{i \notin wait(G)} X_i(G)$.

Definition2 : transition de saut [17]
Une transition de saut est une relation binaire sur un ensemble d'états globaux, elle est notée : $\rightarrow^l$. Pour deux états globaux $G1=((\langle s_i^{G1}\rangle_{i\in I},\langle c_{ij}^{G1}\rangle_{(i,j)\in L})$ et $G2=((\langle s_i^{G2}\rangle_{i\in I},\langle c_{ij}^{G2}\rangle_{(i,j)\in L})$, $G1 \rightarrow^l G2$ existe, ssi il existe un ensemble de transitions appartenant à un ensemble *pleap* de sauts $T \in pleap(G1)$, tel que l'exécution simultanée des transitions de T, fait passer le protocole à l'état global G2.
Soit $\rightarrow^{l*}$ la fermeture transitive et réflexive de $\rightarrow^1$. G2 est dit accessible par sauts depuis G1, ssi $G1 \rightarrow^{l*} G2$. Si G1=S0 (état global initial du protocole), G2 est dit être accessible par saut.
On dénote par L_Π l'ensemble de tous les états accessibles par saut du protocole Π.
Pour une séquence d'ensemble des sauts $\Omega=T_1T_2T_3\ldots T_m$, $G1 \rightarrow^{\Omega*} G2$ dénote l'existence des états globaux $Q^0, Q1, Q^2, \ldots Q^m$, tel que :
$G1=Q^0 \rightarrow^{T1} Q^1 \rightarrow^{T2} Q^2 \ldots.\text{-->}^{Tm} Q^m=G2$.

Dans un graphe de sauts, les nœuds sont des états globaux qui sont accessibles par des sauts. Le calcul des transitions de sauts (pleap) se fait au niveau de chaque état global, en commençant de l'état global initial du système.

IV-3-2) Construction du graphe de sauts

En utilisant les définitions ci-dessus, la construction d'un graphe de sauts ce fait comme suite :
1-On commence la construction depuis l'état global initial du protocole(G^0), en calculant l'ensemble *pleap*(G^0) pour obtenir les premiers arcs du graphe(premiers sauts).
2-Pour chaque nouvel état global obtenu, on refait le calcul de l'ensemble des nouveaux sauts correspondants à chaque état global.
3-Refaire l'étape 2 jusqu'à ne plus avoir d'état global à générer.

IV-3-3) Exemple

Considérons le protocole Π={{P1,P2,P3,P4},{C_{12},C_{23},C_{34}, C_{41},C_{43}}} de la figure 4.1.
On note les transitions du protocole par :

$t_1^1 = (10, -m_{12}, 11)$ $t_2^1 = (20, -m_{23}, 21)$ $t_3^1 = (30, -m_{34}, 31)$ $t_4^1 = (40, -m_{43}, 41)$
$t_1^2 = (10, +m_{41}, 12)$ $t_2^2 = (20, +m_{12}, 22)$ $t_3^2 = (31, +m_{43}, 30)$ $t_4^2 = (41, +m_{34}, 40)$

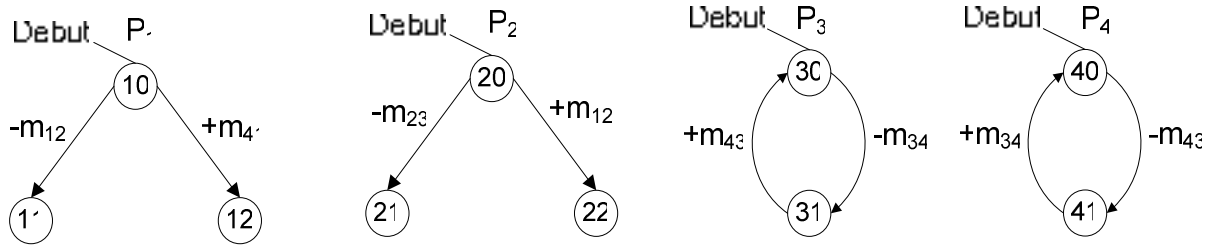


Figure 4.1 : Une spécification de protocole composée de quatre entités [11]

Le graphe de sauts, construit à partir de la spécification de l'exemple de la figure4.1 est donné sur la figure4.2.

La construction du graphe se fait comme suit :
A partir de l'état global initial G^0=(<10,20,30,40>,<ε,ε,ε,ε,ε>) on détermine l'ensemble des transitions exécutables et l'ensemble des transitions potentiellement exécutables du protocole, qui sont respectivement $X(G^0)=\{t_1^1,t_2^1,t_3^1,t_4^1\}$ et $P(G^0)=\{t_1^2,t_2^2\}$.
Les entités P_1 et P_2 disposent de transitions potentiellement exécutables, ce qui donne l'ensemble wait(G^0)={1,2}. Donc, l'ensemble des sauts de l'état initial G^0 sera construit par le produit cartésien des ensembles de transitions exécutables $X_3(G^0)=\{t_3^1\}$ et $X_4(G^0)=\{t_4^1\}$ correspondants aux deux entités P_3 et P_4 respectivement, ce qui donne *pleap*(G^0)=$\{\{t_3^1,t_4^1\}\}$. Donc à partir de G^0 il existe un seul saut qui est réalisé par l'exécution simultané des deux transitions t_3^1 et t_4^1. L'état global résultant de l'exécution de ce saut est G1=(<10,20,31,41>,<ε,ε,m_{34},ε,m_{43}>)
De la même manière et à partir de l'état G1, on calcul l'ensemble *pleap*(G1)=$\{\{t_3^2,t_4^2\}\}$. L'exécution des transitions de cet ensemble, génère l'état global G^0. Puisque on revient sur l'état global initial G^0 la construction du graphe d'accessibilité s'arrête et le résultat obtenu est donné sur la figure4.2 suivante :

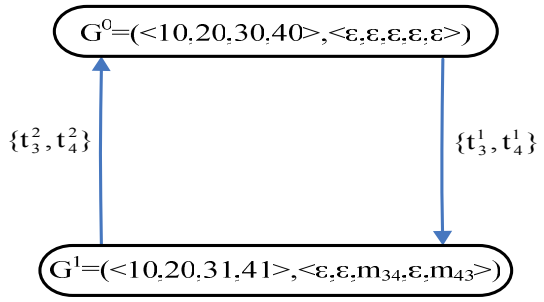


Figure 4.2 : Graphe de sauts correspondant au protocole Π.

IV-4) Une analyse d'accessibilité basée sur des sauts réversibles (Reverse Leaping Reachability Analysis)

Nous proposons dans cette section, une technique de validation de protocole basée sur une approche de retour arrière avec saut que nous avons appelé : **R**everse **L**eaping **R**eachability **A**nalysis (abrégée RLRA). Le principe de cette technique et de faire sortir un ensemble d'états suspects dans une première phase pour en suite confirmer ceux qui pressentent réellement une erreur de spécification dans une deuxième phase. Cette confirmation se fait par la construction d'un graphe de sauts d'une manière réversible. L'état suspect est alors confirmé comme une erreur logique lorsque l'état initial du protocole est atteint à travers le graphe construit.

Dans ce qui suit, nous donnons les différentes définitions qui vont nous permettre de construire le graphe de sauts d'une manière réversible.

Définition3 : état global réversible[14]
Lorsque on parcourt un graphe d'accessibilité d'une manière réversible, l'ensemble des états construit par cette opération est appelé ensemble d'états globaux réversibles.
Un état global réversible d'un protocole Π est noté $G=((\langle s_i^G\rangle_{i\in I},\langle \underline{c}_{ij}^G\rangle_{(i,j)\in L})$ où s_i^G est l'état local de l'entité P_i et $\underline{c}_{ij}^G$ est la séquence de messages **désirés** sur le canal C_{ij} $\forall(i,j)\in L$.

L'opération de retour arrière peut être vue comme une procédure abstraite. A partir d'un état global on monte vers la racine, et dans ce cas-là, les transitions d'envoi de messages vont être traitées comme étant des transitions de réception de messages, et les transitions de réception de messages vont être traitées comme étant des transitions d'envoi de messages.

IV-4-1) Transitions simultanément et réversiblement exécutables

Comme pour la construction des graphes de sauts, nous avons besoin de définir les notions de transitions réversiblement exécutables et de transitions potentiellement et réversiblement exécutables afin de calculer les transitions simultanément et réversiblement exécutables (sauts réversibles).

Transition réversiblement exécutable
Une transition $t=(s_i,\mu,s_i')\in \Delta_i$, est dite réversiblement exécutable à l'état global réversible G, ssi $s_i'=s_i'^G$ ($s_i'^G$ est l'état local de P_i à l'état global G) , et *t* est sous une des deux formes suivantes :
1- *t* est une transition d'envoi de message tel que $\mu=-x$ avec $x\in M_i$ et $Front(\underline{c}_{ij}^G)=x$. ou,
2- *t* est une transition de réception de message et le contenu du canal $\underline{c}_{ji}$ est inférieure à K, i.e. $|C_{ji}|<K$.
L'ensemble des transitions d'envoi et de réception de messages, qui sont réversiblement exécutables à l'état global réversible G, sont notés $X_i^{-R}(G)$ et $X_i^{+R}(G)$ respectivement. L'union de ces deux ensembles est noté $X_i^R(G)= X_i^{-R}(G)\cup X_i^{+R}(G)$. L'ensemble de toutes les transitions du protocole, qui sont réversiblement exécutables à l'état G, est noté :
$X^R(G)= \cup_{i\in I} X^R i(G)$.

Transition potentiellement et réversiblement exécutable
Une transition de réception de message $t=(s_i,+y,s_i')\in \Delta_i$, avec $y\in M_i$, est dite potentiellement et réversiblement exécutable à l'état global réversible G, ssi $s_i=s_i'^G$ et $|\underline{c}_{ji}|=K$; K est la capacité des canaux de communication du système. Une transition d'envoi de message $t=(s_i,-x,s_i')\in \Delta_i$,

avec $x \in M_i$, est dite potentiellement et réversiblement exécutable à l'état global G, ssi $s_i^{'}=s_i^{'G}$ et $\underline{c}_{ij}^{G}=\varepsilon$.
Les ensembles des transitions de réception de messages et des transitions d'envoi de messages de l'entité P_i, qui sont potentiellement et réversiblement exécutables à l'état global G, sont notés $P_i^{R+}(G)$ et $P_i^{R-}(G)$ respectivement. On note $P^R{}_i(G)= P_i^{R+}(G) \cup P_i^{R-}(G)$.

Comme pour la construction d'un graphe de sauts, la construction d'un graphe de sauts réversibles se fait par le calcul des transitions simultanément et réversiblement exécutables, ces transitions doivent être exécutables dans le même état global réversible et doivent appartenir à des entités du protocole qui ne disposent pas de transitions potentiellement et réversiblement exécutables. Un ensemble de telles transitions est appelé sauts réversibles, noté $pleap^R$ (reverse proper leap).
Donc, un saut réversible est un ensemble non vide de transitions : $T \subseteq X^R(G)$, tel que pour toute transition $t,t^{'} \in T$, $act(t) \neq act(t^{'})$ si $t \neq t^{'}$ (i.e T contient au plus une transition réversiblement exécutable de chaque entité). Les différents ensembles T, construits à l'état global réversible G, forment un ensemble de sauts réversibles noté $pleap^R(G)$.

Nous donnons dans ce qui suit la définition formelle de l'ensemble des sauts réversibles ($pleap^R$).

Définition4 : calcul de l'ensemble pleap^R (sauts réversibles)
Soit G un état global réversible.
On définit l'ensemble des entités qui disposent d'une transition potentiellement et réversiblement exécutable à l'état G par : $Wait^R(G)=\{i \in I / X_i^R(G) \neq \varnothing \Rightarrow P^R{}_i(G) \neq \varnothing\}$.

L'ensemble $pleap^R(G)$ est calculé comme suite :

$$pleap^R(G)= \begin{cases} \{\prod_{i \notin wait^R(G)} X_i^R(G)\} \text{ si } Wait^R(G) \subset I \\ \{\{t\}/t \in X^R(G)\} \text{ sinon} \end{cases}$$

Avant de calculer l'ensemble $pleap^R(G)$, on doit calculer l'ensemble $Wait^R$ pour identifier les entités qui disposent de transitions potentiellement et réversiblement exécutables. Si toutes les entités appartiennent à l'ensemble $Wait^R$ ça veut dire qu'il n'y a pas de saut réversible à exécuter à partir de G, et dans ce cas la, on exécute chaque transition de $X^R(G)$ indépendamment des autres (simple transition). Si l'ensemble $Wait^R$ est inclut dans I, l'ensemble de sauts réversibles à exécuter à partir de G ($pleap^R(G)$), sera le produit cartésien des ensembles de transitions réversiblement exécutables de chaque entité($X_i^R(G)$), i.e $pleap^R(G)=\prod_{i \notin wait^R(G)} X_i^R(G)$.

Définition5 : transition de saut réversible
Une transition de saut réversible, est une relation binaire $\rightarrow^R$ sur un ensemble d'états globaux réversibles. Pour deux états globaux réversibles $G1=((\langle s_i^{G1}\rangle_{i \in I}, \langle \underline{c}_{ij}^{G1}\rangle_{(i,j) \in L})$ et $G2=((\langle s_i^{G2}\rangle_{i \in I}, \langle \underline{c}_{ij}^{G2}\rangle_{(i,j) \in L})$, $G1 \rightarrow^R G2$ existe, ssi il existe un ensemble de transitions réversiblement exécutables $T \in pleap^R(G1)$, tel que l'exécution des transitions de T à partir de G1 donne l'état global G2.
La relation de transition $G1 \rightarrow^R G2$ signifie que l'état global G2 est obtenu d'une manière réversible directe depuis l'état global G1.

Soit $\rightarrow^{R*}$ la fermeture transitive et réflexive de $\rightarrow^{R}$, on dit que l'état G2 est accessible par saut réversible de puis G1, ssi $G1 \rightarrow^{R*} G2$. Le chemin qui relie G1 vers G2 est alors appelé : chemin de sauts réversibles de G1 vers G2.

En reprenant l'exemple de la figure4.2, une transition de saut réversible peut être calculée à partir de l'état global réversible G1=(<10,20,31,41>,<ε,ε,$\underline{m_{34}}$,ε,$\underline{m_{43}}$>) comme suite :
L'ensemble des transitions réversiblement exécutables, à partir de G1 est $X^R(G^1)=\{t_3^1, t_4^1\}$. L'ensemble des transitions potentiellement et réversiblement exécutables est $P^R(G^1)=\varnothing$ ce qui fait que l'ensemble des entités qui ont des transitions potentiellement et réversiblement exécutables sera vide : $Wait^R(G^1)=\varnothing$. Donc, l'ensemble des sauts réversibles calculés à partir de l'état G1, $pleap^R(G^1)=\{\{t_3^1, t_4^1\}\}$, est composé d'un seul ensemble contenant deux transitions simultanément et réversiblement exécutables. L'exécution simultanée de ces deux transitions donne l'état global initial du protocole, donc un saut réversible existe entre l'état G1 et G0 (figure4.3).

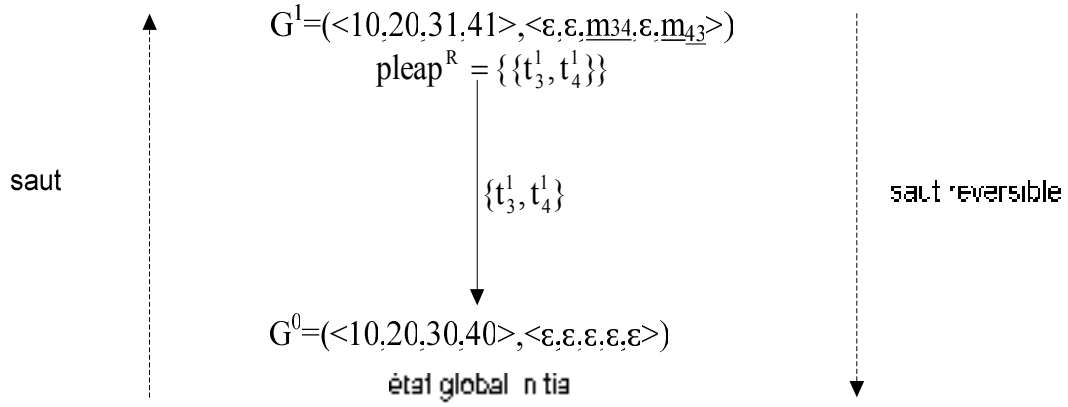


Figure 4.3 : Saut réversible calculé à partir de l'état G1 de l'exemple de la figure4.1.

Les nœuds d'un graphe d'accessibilité, obtenu par l'analyse RLRA, sont des états globaux réversibles qui sont accessibles par saut réversible (*pleap*^{R}). Le calcul des transitions de sauts réversibles se fait au niveau de chaque état global en partant d'un état global donné du protocole.

IV-4-2) Traitement des erreurs d'interblocage

Dans cette section, nous expliquons l'utilisation des sauts réversibles dans la détection des erreurs d'interblocage.

Définition 6 : état suspect d'interblocage[14]
Un état global $G=(\langle s_i^G\rangle_{i\in I}, \langle c_{ij}^G\rangle_{(i,j)\in L})$ est dit un possible état d'interblocage ssi,
$\forall(i,j)\in L : c_{ij}^G=\varepsilon$ et s_i^G est un état de réception de message pour tout $i\in I$.

Définition 7 : état d'interblocage[17]
Un état global $G=(\langle s_i^G\rangle_{i\in I},\langle c_{ij}^G\rangle_{(i,j)\in L})$ est dit état d'interblocage ssi ,
1- $\forall(i,j)\in L : c_{ij}^G=\varepsilon$ et s_i^G est un état de réception de message pour tout $i\in I$,
2- L'état global G est accessible par saut à partir de G0.

Un état global est dit état d'interblocage lorsque tous les canaux de communications dans cet état sont vides et toutes les entités du protocole sont dans un état de réception de messages.

Nous donnons ci-dessous le théorème principal de notre proposition qui nous permettra d'utiliser les chemins de sauts réversibles afin de détecter les erreurs logiques de type interblocage.

Théorème
Un possible état d'inter blocage GS, est confirmé état d'inter blocage ssi, l'état global initial du protocole G0 est accessible par sauts réversibles à partir de l'état suspect GS.

Preuve

Supposons que l'état suspect GS est un état d'interblocage. Par définition 7, GS est accessible par saut depuis l'état global initial G0. Soit $G0 \rightarrow^l G1 \ldots \rightarrow^l Gi \ldots \rightarrow^l GS$ le chemin de sauts de G0 vers GS ($G0 \rightarrow^{l\,*} GS$). En utilisant la définition 5, on peut facilement définir le chemin de sauts réversibles qui lui correspond, $GS=\underline{GS} \rightarrow^R \ldots \rightarrow^R \underline{Gi+1} \rightarrow^R \underline{Gi} \ldots \rightarrow^R \underline{G1} \rightarrow^R \underline{G0}=G0$, où chaque transition entre états réversibles $\underline{Gi+1} \rightarrow^R Gi$ est déclanchée par l'exécution du même ensemble de transitions $T\in pleap(Gi)$ entre les états globaux Gi et Gi+1 du graphe de sauts. Donc G0 est accessible par un chemin de sauts réversibles depuis GS.
Dans l'autre sens, supposons que l'état global G0 est accessible par un chemin de sauts réversibles depuis l'état GS et que le chemin $\underline{GS} \rightarrow^{R\,*} \underline{G0}$ est définit comme suite : $\underline{GS} \rightarrow^R \ldots \rightarrow^R \underline{Gi} \rightarrow^R \underline{Gi+1} \ldots \rightarrow^R \underline{G1} \rightarrow^R \underline{G0}$. En utilisant la définition2, nous construisons le chemin des sauts qui lui correspond : $G0 \rightarrow^l G1 \ldots \rightarrow^l Gi \rightarrow^l Gi+1 \ldots \rightarrow^l GS$ de G0 vers GS, où chaque transition $Gi \rightarrow^l Gi+1$ est déclanchée par l'exécution du même ensemble de transitions $T\in pleap^R(G^{i+1})$. Donc par définition 6 et 7, GS est un état d'interblocage.

On se basant sur le théorème précèdent, la détection des états d'interblocage peut être faite par une vérification de l'accessibilité par sauts réversibles de l'état initial du système en partant d'un état suspect.

IV-5) Algorithme

Afin de détecter les erreurs d'interblocage, notre algorithme agit en deux phases :

1- Dans la première phase, on détermine l'ensemble des états suspects d'interblocage à partir de la spécification du protocole (CAEF ou CAEFC aplatie).

2- Dans la deuxième phase, on confirme parmi les états suspects obtenus dans la première phase, ceux qui présentent réellement une erreur logique, par une vérification de l'existence d'un chemin de sauts réversibles depuis ces états suspects vers l'état initial du protocole.

Pour des raisons de simplifications, nous traitons des protocoles de communication entre deux entités seulement. Notre algorithme peut être facilement généralisé sur une communication entre plusieurs entités.

Chaque entité Pi (i=1,2) est représentée sous forme d'un AEF défini par un quintuple $(\mathbf{S_i,S0_i,S_{fi},M_i,\Delta_i})$:

$\mathbf{S_i}$: l'ensemble de tous les états de l'entité i. Chaque état $\mathbf{s \in S_i}$ est défini par les informations suivantes :(**Nom-état** : Chaine, **Est-Final** :Boulean, **Est-Complexe** :Boulean).

$\mathbf{\Delta_i}$: l'ensemble de toutes les transitions de l'AEF de Pi. Une transition est définie par les informations suivantes :

(**Début_transition** :$s \in S_i$,**Evénement** :enum(+,-), **Message** :$m \in M_i$, **Fin-transition** : $s \in S_i$).

$\mathbf{M_i}$: l'ensemble des messages que peut envoyer ou recevoir l'entité i (Alphabet de l'entité).

IV-5-1) Détection des états suspects (*Phase1)*

Les états suspects d'interblocage sont obtenus par une combinaison, des états locaux de réception de messages, des différentes entités communicantes du protocole (***Définition 6***).
L'algorithme du calcul des états suspects est donné sur la figure 4.4 suivante :

Entrée : protocole Π (P_1, P_2).
$P_i=(S_i,S0_i,E_{fi},M_i,\Delta_i)$;$i \in \{1,2\}$.
Sortie : ensemble des états suspects :**Suspect**.

```
1 Suspect= ∅ ;
2 Pour chaque état de réception de message s1 dans S1
3   Pour chaque état de réception de message s2 dans S2
4         Si s1 et s2 sont des états non finaux faire
5           GS=(e1,e2,∅,∅) ; //générer un état suspect
6           Suspect= Suspect ∪ {GS} ;
7         Fsi;
8   Fpr ;
9  Fpr ;
10 Retourne Suspect ;
11 Fin ;
```

Figure 4.4 : Phase de détection des états suspects

Le test de la ligne 4 de l'algorithme sert à écarter les états locaux qui sont finaux car un état global final ne peur être état d'interblocage.

IV-5-2) Confirmation des erreurs d'interblocage (*Phase2)*

Afin de confirmer l'existence d'une erreur dans un état suspect GS, l'algorithme effectue une recherche d'un chemin de sauts réversibles vers l'état initial du protocole.

L'algorithme est donné sur la figure 4.5 suivante :

```
Entrée : - Protocole Π (P1, P2).
         - Pi=(Si,S0i,Efi,Mi,Δi);i∈{1,2}.
         - Etat global suspect GS:(s1, s2, ∅, ∅)  //s1∈ S1, s2∈S2
Sortie : Confirmation ou non de l'erreur d'interblocage dans l'état suspect GS.

 1 Explored=∅ ; //Etats déjà explorés
 2 Open={GS} ;    // Etats qui restent à explorer
 3 Tant que Open ≠ ∅ faire
 4       Prendre un état G de Open;
 5        Open= Open-{G} ;
 6       Explored= Explored ∪ {G} ;
 7      Si G=G0 alors
 8         Si GS.s1=complexe et GS.s2=complexe alors
 9           Retourne ''GS confirmé comme erreur d'interblocage complexe'';
10         Sinon si GS.s1=complexe ou GS.s2=complexe alors
11          Retourne ''GS confirmé comme erreur d'interblocage hybride'';
12      Sinon Retourne ''GS confirmé comme erreur d'interblocage simple'';
13     Fsi ; //fin si
14    Sinon calculer pleap^R(G) ; // calcul des sauts réversibles à partir de G
15         Pour tout T dans pleap^R(G)
16             Générer un état G' par l'exécution simultané des transitions de T ;
17             Si G' ∉ Explored alors //état non déjà traité
18                  Open=Open ∪ {G'} ;
19              Fsi ;
20         Fpr; //fin boucle pour
21    Fsi ;//fin sinon
22  FinTq ;
23 Retourne ''GS n'est pas une erreur de protocole'' ;
24 Fin ;
```

Figure 4.5 : Phase de confirmation des erreurs d'interblocage.

L'ensemble **Explored** est utilisé pour marquer les états globaux qui sont déjà traités (vérifiés). Chaque nouvel état global n'appartenant pas à **Explored** sera mis dans **Open** afin de le traiter dans la prochaine itération (ligne 16-17-18).

A partir de chaque état global G' (non traité), nous calculons tous les ensembles des transitions simultanément exécutable (T) de cet état global, par l'utilisation de la **définition 4** (ligne 14). Pour chaque **T** obtenu et qui est un saut réversible, nous générons un nouvel état global. L'algorithme continue son exécution jusqu'à ce que soit l'état initial est atteint, est dans ce cas la l'erreur logique est confirmée dans l'état suspect GS, soit il ne reste plus d'état à explorer et donc l'erreur logique est écartée.
Les lignes de code de 7 à 13 permettent d'identifier le type de l'erreur détectée (simple, hybride ou complexe), selon la nature des états locaux composant l'état de l'erreur.

IV-6) Complexité de l'algorithme

Le résultat de calcul de la complexité au pire cas des deux phases de l'algorithme est le suivant :

1. La complexité en espace de la première phase de calcul d'état suspect est $O(n^2)$. La complexité en temps d'exécution est $O(n^2)$; n étant la taille du plus grand ensemble d'états des deux entités communicantes du protocole.
2. La deuxième phase de l'algorithme utilise la récursivité pour vérifier l'existence d'un chemin entre l'état suspect et l'état initial du graphe. La complexité de tel algorithme est $O(b^p)$ en temps et O(bxp) en espace; b est le facteur de branchement et p est la profondeur.

La complexité de notre algorithme dépend fortement du nombre d'états suspects détectés ainsi que l'espace des états à parcourir avant la fin de l'analyse. En utilisant un parcourt du graphe de sauts, notre algorithme permet de diminuer le facteur de branchement ainsi que la profondeur de calcul et cela à travers l'exécution simultané de plusieurs transitions dans une même branche.

IV-7) Conclusion

Dans ce chapitre, nous avons proposé une nouvelle technique de validation de protocole contre les erreurs d'interblocage. Notre technique est basée sur le principe de retour arrière avec un parcours réversible des chemins du graphe de sauts. L'avantage d'utiliser des graphes de sauts est d'éviter le parcours des chemins qui ne sert pas dans l'analyse et donc réduire la taille du graphe d'accessibilité du protocole tout en conservant la capacité de détecter toutes les erreurs possibles du protocole.

Les tests et validation de notre proposition feront l'objet du chapitre suivant, où nous montrons l'efficacité de notre technique dans la détection des erreurs d'interblocage.

Chapitre V

Implémentation, tests et comparaison

Pour être analysée, une spécification de protocole doit être sous forme d'un fichier XML. Ce fichier doit contenir la spécification de toutes les entités communicantes du protocole. Cette spécification doit être sous forme d'un modèle de CAEFC aplati ou bien sous forme d'un modèle de CAEF. A partir de la spécification XML, le programme récupère les différentes informations du protocole dans des structures de données à fin de procéder à l'analyse.

Dans ce chapitre, nous donnons l'implémentation et les tests de notre algorithme. Au premier point, nous donnons la structure que doit avoir le fichier XML qui va contenir la spécification. Dans un deuxième point, nous montrons la capacité de notre algorithme à détecter les différents types d'erreurs d'interblocage, c'est-à-dire les interblocages simples , hybrides et complexes. Dans un dernier point, nous testons et comparons l'efficacité de notre algorithme par rapport à une autre technique de la même famille, qui est l'analyse d'accessibilité réversible (RRA)[14].

V-1) Spécification XML

La spécification d'un protocole, modélisé sous forme de CAEFC ou CAEF, doit respecter un format XML préalablement définit. La DTD (Document Type Definition) que doit respecter les fichiers XML à utiliser dans l'analyse est donnée sur la figure 5.1.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
< !DOCTYPE CAEF [
<!ELEMENT CAEF (Automate+) >
<!ELEMENT Automate (Etat+) >
<!ATTLIST  Automate nom CDATA #REQUIRED>
<!ELEMENT Etat (Transition+)>
<!ATTLIST  Etat nom CDATA #REQUIRED>
<!ATTLIST  Etat EstFinale CDATA #REQUIRED>
<!ATTLIST  Etat EstComplexe CDATA #REQUIRED>
<!ELEMENT Transition (Evenement, Message, Vers)>
<!ELEMENT Evenement (#PCDATA)>
<!ELEMENT Message (#PCDATA)>
<!ELEMENT Vers (#PCDATA)>
```

Figure 5.1 :DTD à utiliser pour la spécification XML.

La balise <CAEF> peut contenir un ou plusieurs automates de spécification. Chaque automate est décrit par une balise <Automate>, où est donné l'ensemble des différents états qui le constituent ainsi que l'ensemble de transitions qui relient entre ces différents états.

Dans un automate complexe, l'attribut ***EstComplexe*** d'un état prend la valeur *vrai*, lorsque cet état fait partie d'un automate interne. Les états simples du niveau le plus haut dans la représentation de l'automate, prennent toujours la valeur *faux* pour leur attribut ***EstComplexe***. L'attribut ***EstFinal,*** prend la valeur *vrai,* seulement pour les états qui sont finaux dans l'automate aplati.

Un exemple de spécification XML est donné sur la figure5.3. Cette spécification correspond à l'AEFC de la figure5.2. L'exemple est un AEFC contenant un état interne qui est lui aussi composé de trois états. L'état 0 et 5 sont respectivement l'état initial et l'état final de l'automate.
Dans tous les exemples utilisés dans cette section l'états 0 indique l'état initial de l'automate.

.

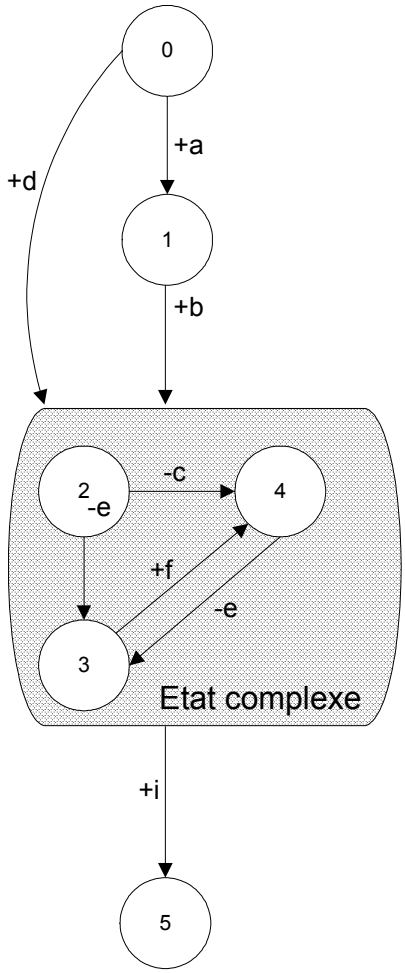


Figure 5.2 : Exemple d'AEFC [30].

Pour valider un protocole, le fichier XML contenant sa spécification, doit être passé comme paramètre à la classe *parser* (classe Java), qui fait analyser son contenu pour en récupérer les données de spécification. Ces données sont en suite passées au programme qui implémente notre algorithme de validation afin de procéder à l'analyse du protocole. Le programme est implémenté en langage Java.
A la fin de l'exécution les différentes erreurs d'interblocage détectées sont affichées en sortie.

```
<CAEF>
<Automate nom="M1">
<Etat nom="0" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>a</Message>
<Vers>1</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>d</Message>
<Vers>2</Vers>
</Transition>
</Etat>
<Etat nom="1" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>b</Message>
<Vers>2</Vers>
</Transition>
</Etat>
<Etat nom="2" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>-</Evenement>
<Message>e</Message>
<Vers>3</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>c</Message>
<Vers>4</Vers>
</Transition>
</Etat>
<Etat nom="3" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>+</Evenement>
<Message>f</Message>
<Vers>4</Vers>
</Transition>
</Etat>
<Etat nom="4" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>-</Evenement>
<Message>e</Message>
<Vers>3</Vers>
</Transition>
</Etat>
<Etat nom="5" EstFinale="vrai" EstComplexe="faux">
</Etat>
</Automate>
</CAEF>
```

Figure 5.3 : Spécification XML de l'automate de l'exemple de la figure5.2.

V-2) Exemples de tests

Dans ce qui suit, nous testons la capacité de notre algorithme à détecter les différents types d'erreur et cela à travers des exemples choisis de façon à ce que chaque exemple contient un type d'erreur.

V-2-1) Scénario d'interblocage simple

Une erreur d'interblocage est dite de type simple, lorsque toutes les entités communicantes du protocole sont dans leurs états simples. Une CAEF qui contient un scénario d'interblocage simple est donnée sur la figure5.4. Le fichier de spécification XML de cet exemple est donné dans l'annexe A.

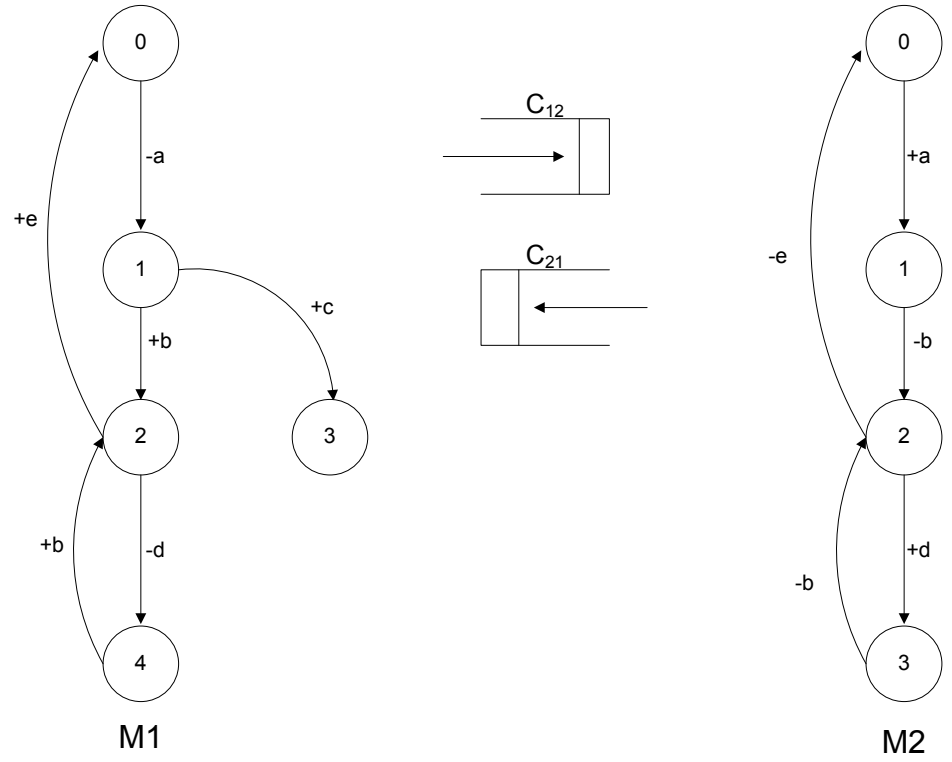


Figure 5.4 : Un scénario d'interblocage simple.

Les transitions du protocole sont :
$t_1^1 = (0,-a,1)$, $t_1^2 = (1,+b,2)$, $t_1^3 = (1,+c,3)$, $t_1^4 = (2,+e,0)$, $t_1^5 = (2,-d,4)$, $t_1^1 = (4,+b,2)$.
$t_2^1 = (0,+a,1)$, $t_2^2 = (1,-b,2)$, $t_2^3 = (2,-e,0)$, $t_2^4 = (2,+d,3)$, $t_2^5 = (3,-b,2)$.
Au début de la communication, les deux entités M1 et M2 sont dans leurs états initiaux à l'état global <0,0,ε,ε>. M1 envoie un message 'a' et passe à l'état 1 en suite M2 reçoit le message 'a' est passe à l'état 1. L'état global du protocole devient <1,1,ε,ε>, cet état représente un état d'interblocage, comme tous les états des entités du protocole sont des états de réception de message avec des canaux de communication vides. L'interblocage est dit simple puisque les deux entités sont dans des états simples.

Le résultat de l'exécution de notre programme sur l'exemple précédent est illustré sur la figure5.6. Le programme récupère les données de l'analyse depuis le fichier de spécification XML, en suite la première phase est entamée pour faire sortir les états globaux suspects d'interblocage par une opération de combinaison entre les états des entités qui ne disposent que de transitions de réception de messages. Pour l'entité M1, l'ensemble des états de réception est composé des états 1 et 4. l'ensemble des états de réception de M2 est composé des états 0 et 1, donc le résultat de combinaison entre les deux ensembles donne l'ensemble des états globaux suspects suivant :{(1,0,ε,ε) , (1,1,ε,ε), (4,0,ε,ε), (4,1,ε,ε)}.
La deuxième phase est exécutée sur l'ensemble des états suspects précédent pour confirmer les états qui présentent réellement un interblocage. Parmi les quatre états suspects, seul l'état global (1,1,ε,ε) est confirmé état d'interblocage car un chemin de sauts réversibles est trouvé depuis cet état vers l'état global initial du protocole. Le graphe d'accessibilité obtenu est illustré sur la figure5.5 suivante :

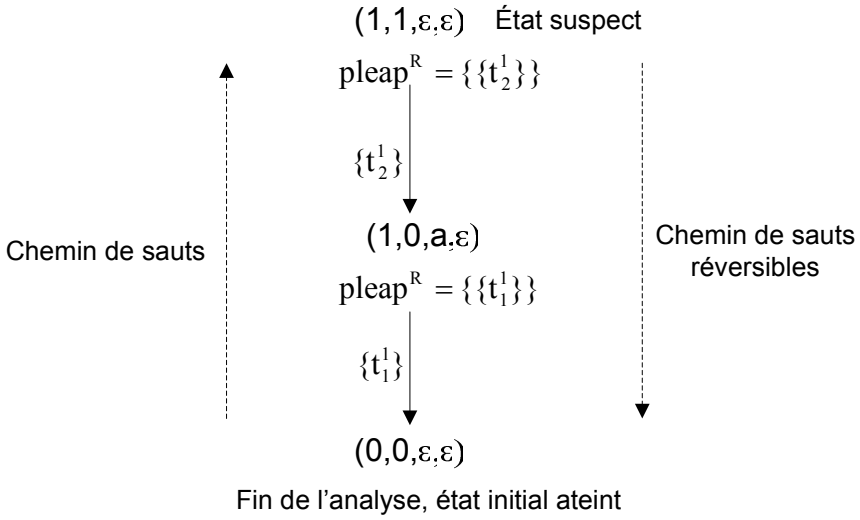


Figure 5.5 : Chemin de sauts réversibles obtenu à partir de l'état suspect (1,1,ε,ε).

A partir de l'état global (1,1,ε,ε) l'ensemble des transitions de sauts réversibles $pleap^R$ est composé d'un seul ensemble qui contient une seule transition t_2^1 et qui est la seule transition exécutable dans cet état global. L'état global résultant de l'exécution de t_2^1 est (1,0,a,ε). De même, à partir de cet état global l'ensemble $pleap^R$ est calculé et le résultat obtenu est un ensemble composé d'une seule transition t_1^1. L'exécution de cette transition donne l'état global (0,0,ε,ε). Puisque cet état global est l'état global initial du protocole, l'analyse se termine et l'état suspect est donc confirmé un état d'interblocage.
L'exécution de l'algorithme sur le reste des états suspects donne des chemins qui ne permettent pas d'accéder à l'état global initial et par conséquent ces états ne sont pas confirmés comme des erreurs d'interblocage.

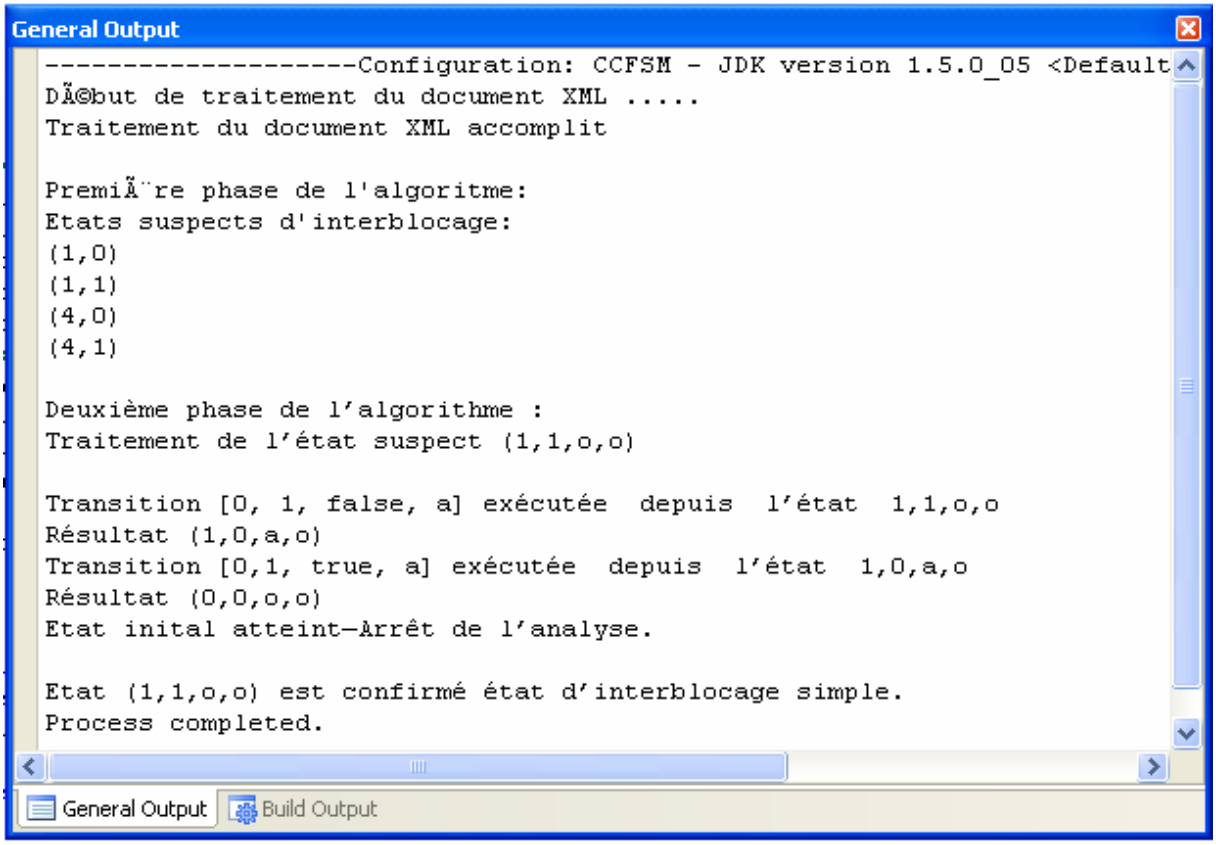


Figure 5.6 : Résultat de sortie pour l'exemple de la figure 5.4.

V-2-2) Scénario d'interblocage hybride

Un exemple de CAEFC contenant une erreur d'interblocage hybride est illustré sur la figure5.7. Le fichier de spécification XML correspondant est donné dans l'annexe A.

Les transitions de l'entité M1 sont:

$t_1^1 = (0,-a,1)$, $t_1^2 = (1,-b,2)$, $t_1^3 = (1,+c,3)$, $t_1^4 = (2,+a,1)$, $t_1^5 = (2,-d,4)$, $t_1^6 = (2,-d,4)$, $t_1^7 = (4,+e,5)$, $t_1^8 = (5,+f,4)$, $t_1^9 = (5,-h,7)$, $t_1^{10} = (5,+g,6)$, $t_1^{11} = (6,+i,6)$, $t_1^{12} = (6,-j,8)$, $t_1^{13} = (7,-d,4)$, $t_1^{14} = (7,-j,8)$.

Les transitions de l'entité M2 :

$t_2^1 = (0,+a,1)$, $t_2^2 = (1,-c,0)$, $t_2^3 = (1,+b,2)$, $t_2^4 = (2,-a,1)$, $t_2^5 = (2,+d,3)$, $t_2^6 = (3,+e,4)$, $t_2^7 = (4,-h,2)$, $t_2^8 = (4,-f,3)$, $t_2^9 = (4,-g,5)$, $t_2^{10} = (5,-i,5)$, $t_2^{11} = (5,+j,6)$.

Dans cet exemple, le scénario d'interblocage apparaît entre l'état 4 de l'automate M1 qui est dit complexe (puisqu'il appartient à l'état complexe S1) et entre l'état simple 3 de l'automate M2.

Le chemin du graphe qui permet d'atteindre l'état d'interblocage hybride (4,3,ε,ε) est construit comme suit :

Au début de la communication, les deux entités sont dans leurs états initiaux dans l'état global (0,0,ε,ε). M1 commence la conversation par l'envoi du message 'a' vers M2 sur le canal C_{12} ;

son état local devient 1 (transition t_1^1). M2 récupère le message envoyé par M1 depuis le canal C_{12} et passe à l'état 1 (transition t_2^1). Après l'exécution de ces deux transitions, l'état global du protocole devient (1,1,ε,ε) . M1 envoie un autre message 'b' à partir de l'état 1 et passe à l'état 2 (transition t_1^2), M2 reçoit le message 'b' et passe lui aussi à l'état 2 (transition t_2^3). L'état global du protocole devient (2,2,ε,ε). A partir de cet état global, M1 peut envoyer un 'd' pour passer à l'état 4 (transition t_1^5) et M2 reçoit le message 'd' pour passer à l'état 3 (transition t_2^5). Le protocole se retrouve à l'état global (4,3,ε,ε). Au niveau de cet état global, les deux entités M1 et M2 sont toutes les deux dans des états de réception de messages, avec un contenu de canal vide, ce qui présente un état d'interblocage dans la spécification du protocole.

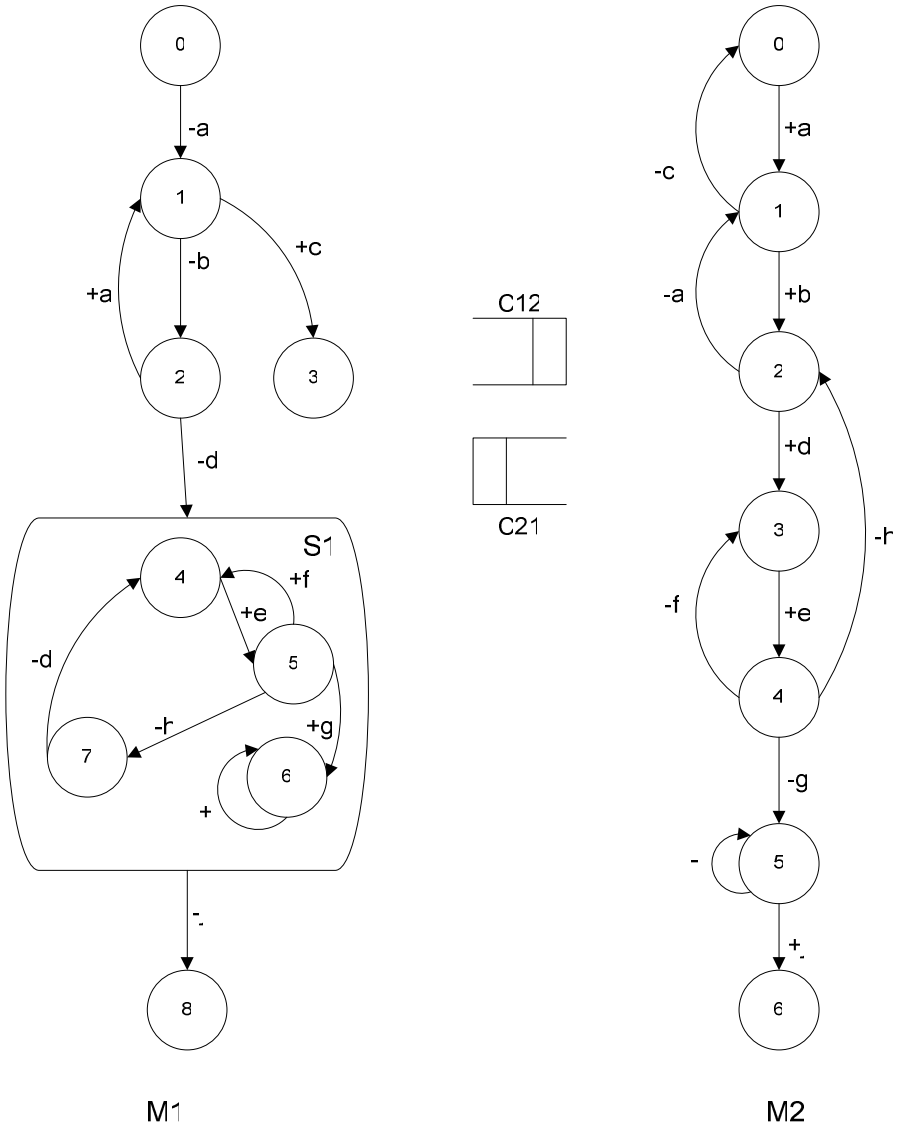


Figure 5.7 : scénario d'interblocage hybride[30].

La détection de l'état d'interblocage (4,3,ε,ε) par l'application de notre algorithme, se fait par la confirmation de cet état comme erreur d'interblocage. Le chemin de sauts réversibles obtenu à partir de l'état suspect (4,3,ε,ε) est illustré sur la figure5.8.

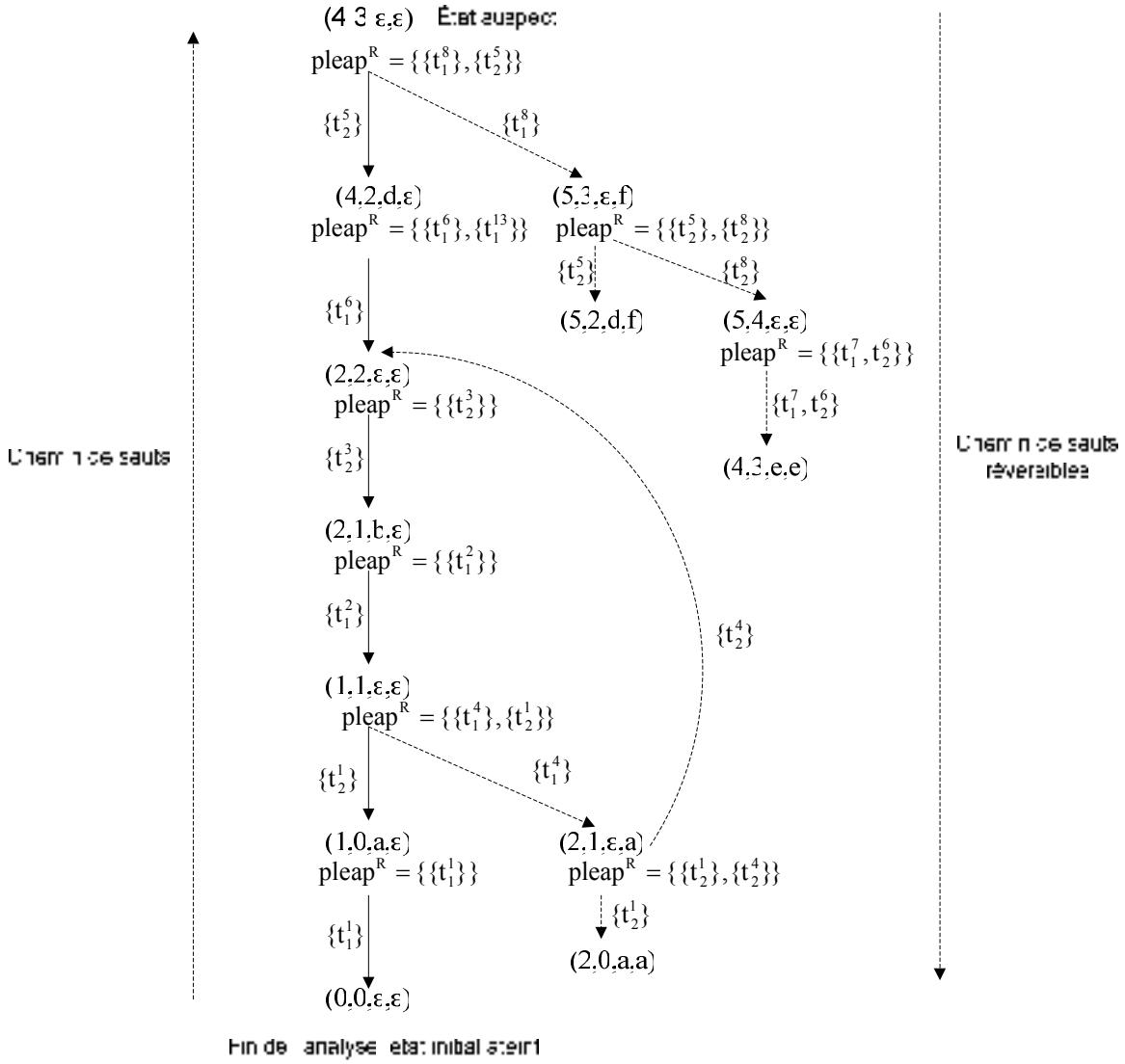


Figure 5.8 : Graphe de sauts réversibles de l'état (4,3,ε,ε).

Les flèches discontinues représentent des chemins qui ne mènent pas vers la racine. Nous avons seulement représenté les chemins de sauts qui permettent d'atteindre la racine. Lorsque on parcourt les chemins obtenus dans l'autre sens, c'est-à-dire à partir de la racine vers l'état suspect, on obtient les chemins de sauts du graphe.

Une prise d'écran du résultat de sortie de l'exécution de notre programme est donnée sur la figure5.9.

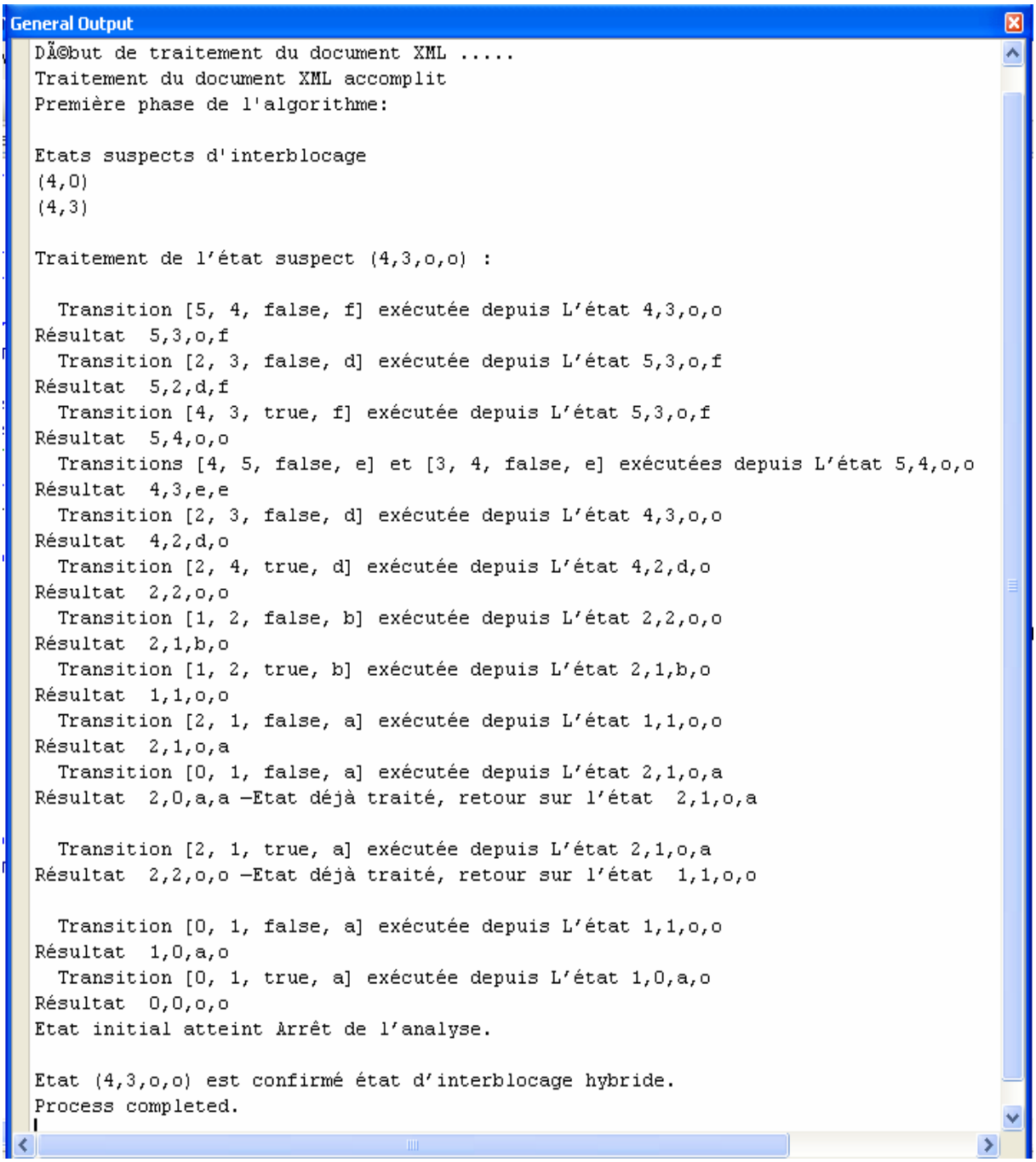


Figure 5.9 : Aperçu du résultat d'exécution de l'exemple de la figure5.7

V-2-3) Scénario d'interblocage complexe

Une erreur d'interblocage est dite complexe, lorsque les états du protocole qui sont en interblocage sont tous des états complexes. Sur la figure 5.10, le scénario d'interblocage complexe apparaît dans les états complexes S2 et S3. L'exécution du protocole se déroule sans erreur jusqu'à atteindre l'automate interne S2 dans la modélisation de M1 et l'automate interne S3 dans la modélisation de M2. Dans ces états internes, M1 et M2 se trouvent dans les états 8 et 6 respectivement. M1 commence par envoyer un message 'k' et passe à l'état 9 et M2 reçoit le message 'k' sur l'état 6 et passe à l'état 7. L'exécution continue jusqu'à ce que

M1 et M2 se retrouvent dans leurs états 8 et 11 respectivement. L'état global du protocole devient (11,8,ε,ε). Dans cet état global, M1 et M2 attendent la réception d'un message 'q' pour pouvoir continuer leur exécution. De plus, avec un contenu de canal vide, le protocole est dit en interblocage. Comme le protocole se trouve dans des états complexes, l'erreur d'interblocage est dite complexe.

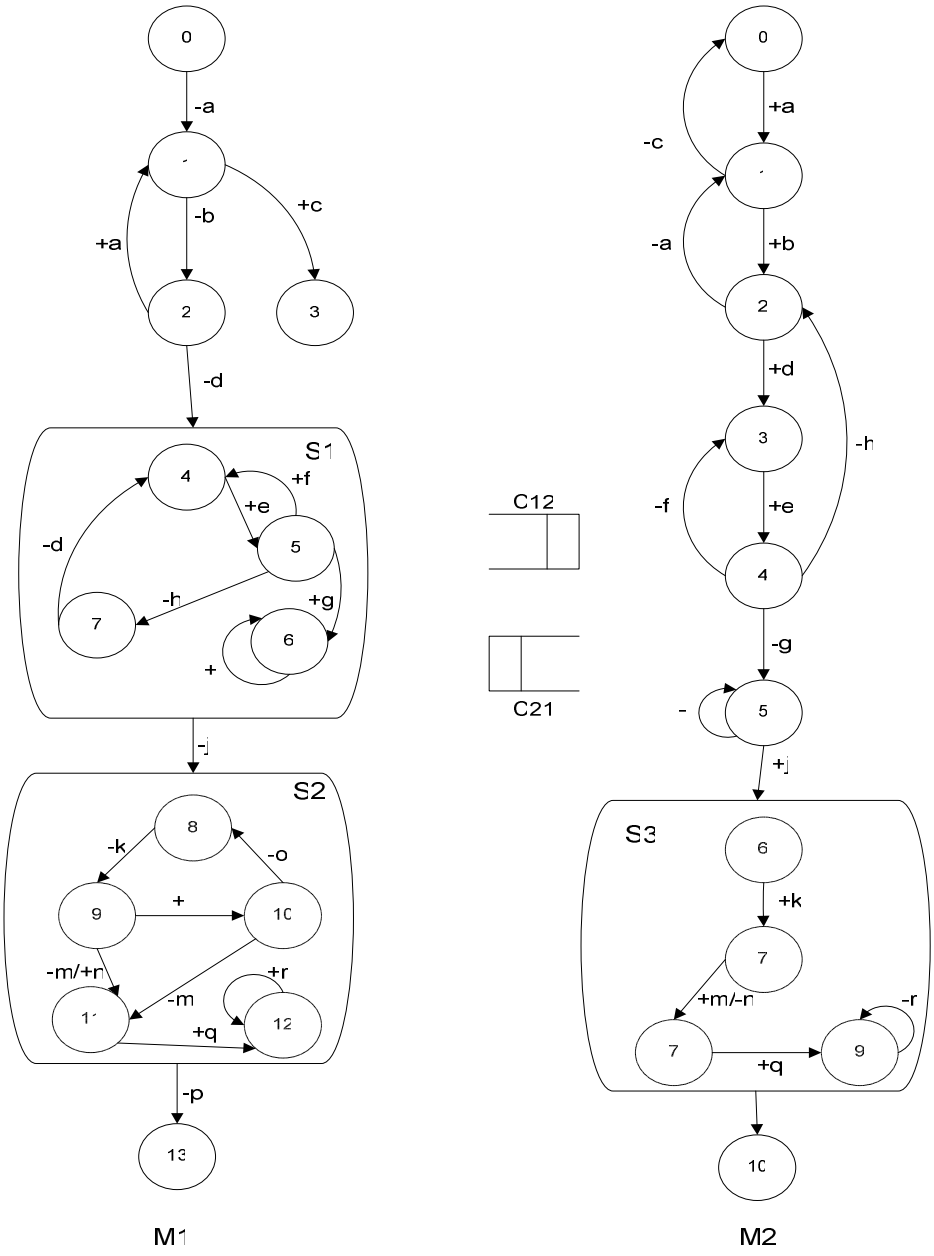


Figure 5.10 : scénarios d'interblocage complexe[30].

Le résultat de l'exécution de notre algorithme sur cet exemple est donné sur la figure5.11. L'état global (11,8,ε,ε) est détecté comme étant un état suspect d'interblocage durant la première phase. Cet état est confirmé comme erreur d'interblocage dans la deuxième phase car l'état initial global du protocole est atteint à travers un chemin de sauts réversibles.

Figure 5.11 : Résultat d'exécution de l'algorithme sur l'exemple de la figure5.10.

V-3) Comparaison

Pour montrer l'efficacité de notre technique, nous avons choisi de la comparer avec une technique de la même famille, c'est-à-dire une technique qui utilise le principe de retour arrière. La technique choisie est l'analyse d'accessibilité réversible (Reverse Reachability analysis) RRA. Cette technique a prouvé son efficacité par rapport à plusieurs d'autres techniques de validation.
La comparaison se fait sur la taille du graphe d'accessibilité à générer, afin de valider le protocole. La taille d'un graphe d'accessibilité est mesurée par rapport au nombre d'états globaux générés et par rapport au nombre de transitions exécutées.

Nous avons fait notre comparaison sur les mêmes exemples utilisés pour valider la technique RRA. Pour chaque exemple, nous avons compté le nombre d'états globaux générés et le nombre de transitions exécutées par chaque technique. De plus, pour avoir une bonne comparaison, nous avons fait varier la capacité des canaux de communication entre 1 et 5.

Le premier exemple modélise une CAEF[11] (Fig5.12). L'exemple contient quatre cas suspects d'interblocage : (0,1,ε,ε), (0,3,ε,ε), (2,1,ε,ε) et (2,3,ε,ε).Tous ces états suspects ne sont pas confirmés comme des erreurs d'interblocage est donc le protocole ne contient pas d'erreur.

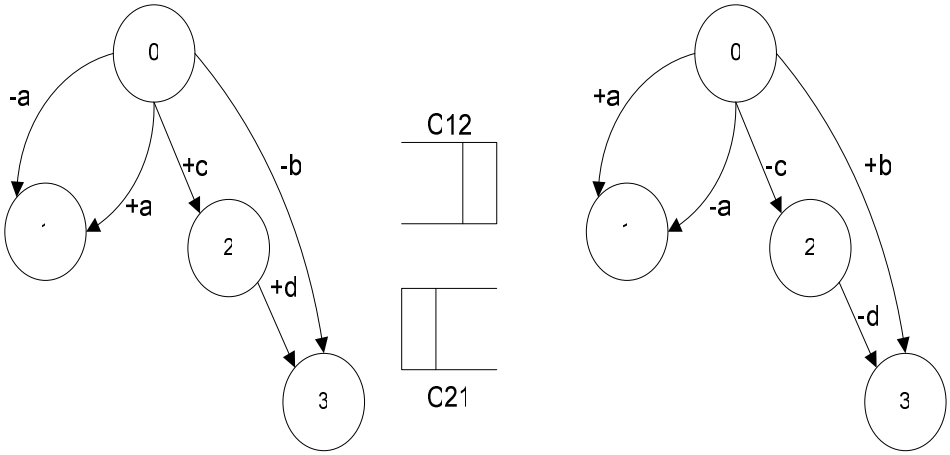


Figure 5.12 **:** Premier exemple de comparaison[11].

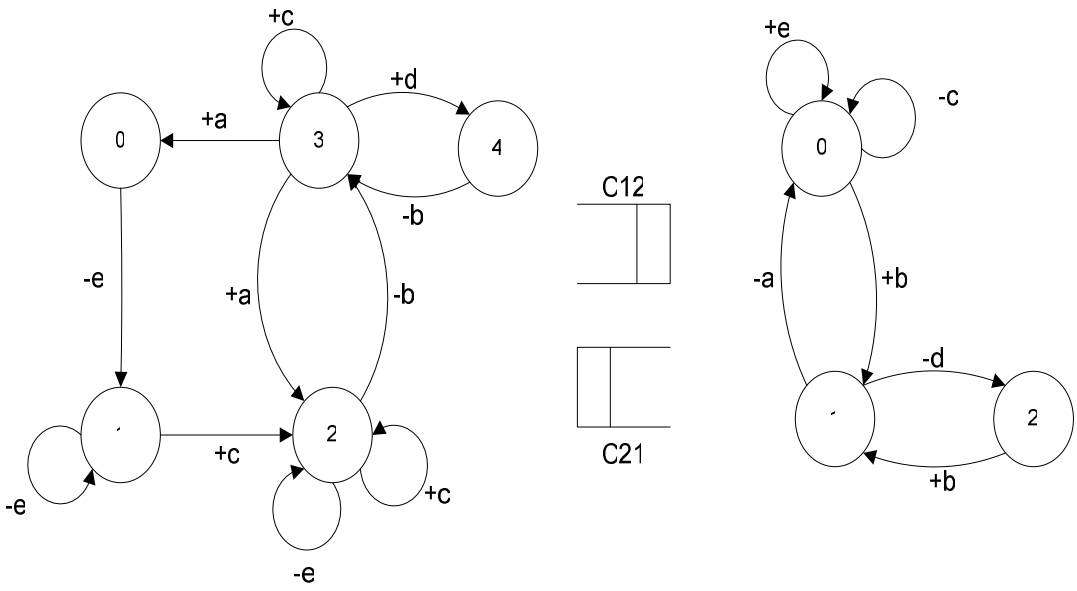


Figure 5.13 : Deuxième exemple de comparaison[8].

Dans le deuxième exemple[8] (Fig5.13), un seul état suspect est détecté : (2,3,ε,ε). Cet état n'est pas confirmé état d'interblocage. Donc le protocole ne contient pas d'erreur.

Le troisième exemple de comparaison, sur la figure5.14 de la page suivante, modélise la procédure d'appel de connexion / Déconnexion, utilisée dans le protocole X .25. Cet exemple contient un seul état suspect qui est l'état (2,3,ε,ε). Le protocole ne présente pas d'erreur d'interblocage.

Le quatrième exemple de test est donné sur la figure5.15. Pour cet exemple, un seul état suspect est détecté et cet état n'est pas confirmé erreur d'interblocage.

Le dernier exemple de comparaison (figure5.16) contient les quatre états suspects suivants : (2,1,ε,ε), (2,2,ε,ε), (3,1,ε,ε) et (3,2,ε,ε). Dans cet exemple, le seul état suspect confirmé comme erreur d'interblocage est l'état (2,2,ε,ε).

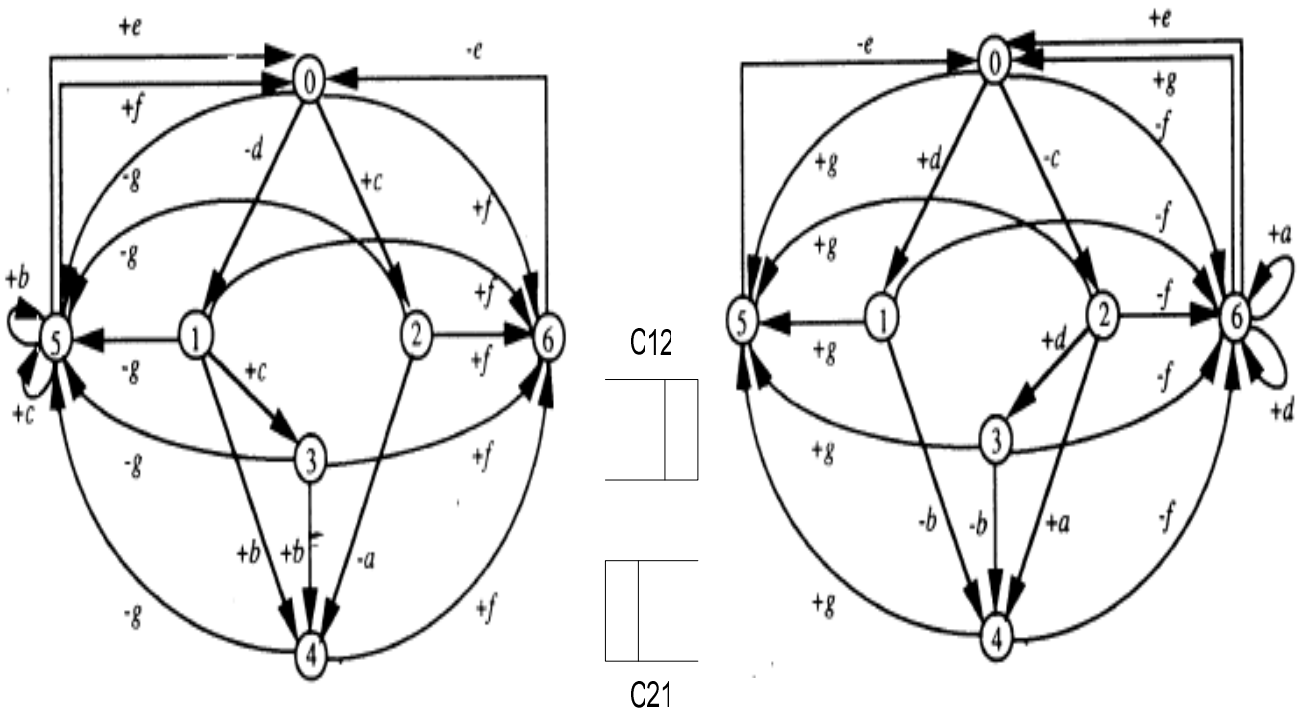


Figure 5.14 : Troisième exemple de comparaison.

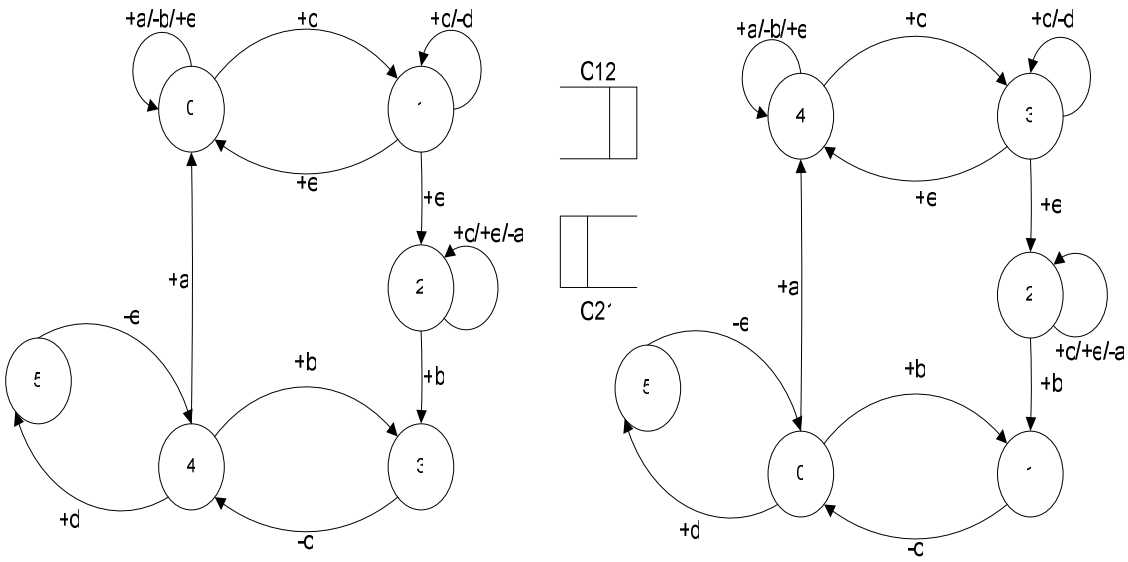


Figure 5.15 : Quatrième exemple de comparaison.

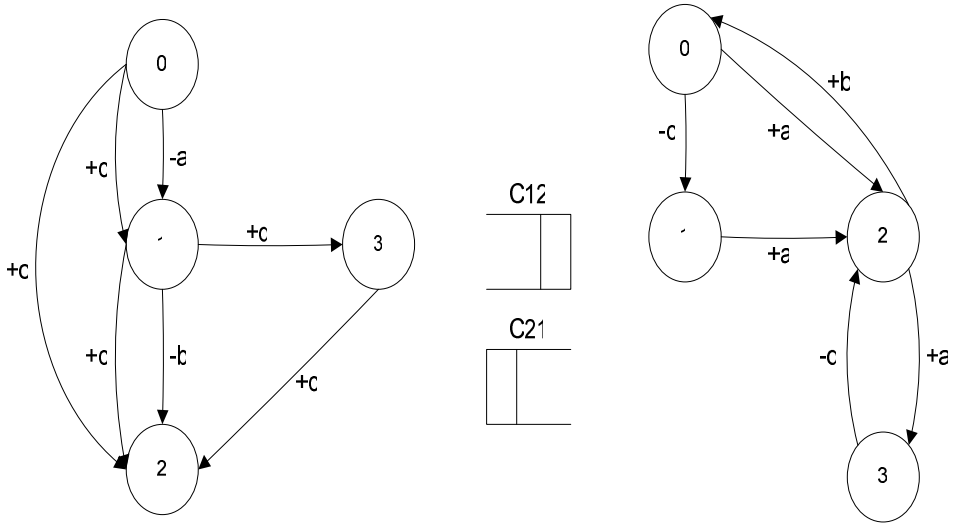


Figure 5.16 : Cinquième exemple de comparaison.

Le résultat obtenu par l'application des deux techniques, par rapport au nombre d'états globaux générés et au nombre de transitions exécutées est donné sur les tableaux 5.1 et 5.2 respectivement.

Capacité des canaux (C12=C21)	Exemples de comparaison														
	Exemple 1			*Exemple 2*			*Exemple 3*			*Exemple 4*			*Exemple 5*		
	RRA	**RLRA**	**DIFF** RRA-RLRA	**RRA**	**RLRA**	**DIFF** RRA-RLRA	**RRA**	**RLRA**	**DIFF** RRA-RLRA	**RRA**	**RLRA**	**DIFF** RRA-RLRA	**RRA**	**RLRA**	**DIFF** RRA-RLRA
1	6	6	0	2	2	0	1	1	0	9	9	0	54	46	8
2	6	6	0	3	2	1	1	1	0	49	41	8	249	192	57
3	6	6	0	4	2	2	1	1	0	225	201	24	459	457	2
4	6	6	0	5	2	3	1	1	0	961	905	56	726	748	22
5	6	6	0	6	2	4	1	1	0	3969	3849	120	700	523	177

RRA: Reverse Reachability ; **RLRA** : Reverse Leaping Reachability Analysis; **DIFF** : Différence (RRA-RLRA).

Tableau 5.1 : Le nombre d'états globaux générés par les techniques RRA et RLRA.

En comparant le résultat obtenu par rapport au nombre d'états globaux générés, on voie clairement que la technique RLRA génère moins d'états que la technique RRA et cela dans la plupart des exemples de tests. Pour l'exemple 2, avec RLRA on obtient un nombre d'états constant quelque soit la taille des canaux de communication alors qu'avec la technique RRA le nombre d'états augmente avec la taille de ces derniers. Pour l'exemple 4 et 5 notre technique permet d'avoir un nombre d'états moins que ce lui obtenu par l'application de la technique RRA et cela quelque soit la taille des canaux de communication. Si l'application des deux techniques sur l'exemple 1 et 3 donne le même résultat, c'est parce que ces deux exemples sont très simples et ne contiennent pas une modélisation complexe, qui permet d'avoir des exécutions simultanées de transitions et par conséquent il n'y a pas de réductions possibles sur leur graphe d'accessibilité.

Capacité des canaux (C12=C21)	Exemples de comparaison														
	Exemple 1			*Exemple 2*			*Exemple 3*			*Exemple 4*			*Exemple 5*		
	RRA	**RLRA**	**DIFF** RRA-RLRA	**RRA**	**RLRA**	**DIFF** RRA-RLRA	**RRA**	**RLRA**	**DIFF** RRA-RLRA	**RRA**	**RLRA**	**DIFF** RRA-RLRA	**RRA**	**RLRA**	**DIFF** RRA-RLRA
1	2	2	0	1	1	0	0	0	0	12	12	0	58	43	15
2	2	2	0	2	1	1	0	0	0	84	60	24	384	257	127
3	2	2	0	3	1	2	0	0	0	420	348	72	631	571	60
4	2	2	0	4	1	3	0	0	0	1860	1692	168	986	984	2
5	2	2	0	5	1	4	0	0	0	7812	7452	360	919	638	281

RRA :Reverse Reachability ; **RLRA** : Reverse Leaping Reachability Analysis; **DIFF** : Différence (RRA-RLRA).

Tableau 5.2 : Le nombre de transitions exécutées par les techniques RRA et RLRA.

Le tableau de comparaison de nombre de transitions générées montre clairement la réduction apportée par la technique RLRA sur la plupart des exemples de test. Si on prend l'exemple 4 on obtient une réduction de 360 transitions pour un canal de capacité 5. La même chose pour l'exemple 5 où la réduction atteint jusqu'à 281 transitions.

Après comparaison des résultats obtenus dans les tableaux 1 et 2, nous pouvons conclure que la techniques de validation RLRA, que nous avons proposé, donne un graphe d'accessibilité réduit par rapport à celui obtenu par l'application de la technique RR dans la majorité des exemples et dans le cas où aucune réduction n'est possible la taille du graphe reste la même.

V-4) Conclusion

Dans ce chapitre, nous avons donné l'implémentation et les tests de la technique de validation de protocole RLRA. Nous avons montré la capacité de notre technique à détecter toutes les erreurs d'interblocage que peu contenir une spécification de protocole. La technique permet aussi, pour chaque erreur d'interblocage détectée, de déterminer son type (simple, hybride ou complexe).
Pour montrer l'efficacité de la technique proposée, nous l'avons comparé avec la technique d'accessibilité réversible (RRA), qui a déjà prouvé son efficacité de puis longtemps. La comparaison c'est faite sur les mêmes exemples utilisés pour valider cette dernière et cela pour avoir une bonne comparaison. Le résultat de comparaison a montré que technique RLRA permet d'avoir un graphe d'accessibilité de protocole plus réduit.

Conclusion générale

Dans un modèle de service web à base d'agent, l'interaction entre agents(services) doit être implémentée d'une manière dynamique pour permettre une meilleur interopérabilité. Cette interaction entre agent est définit par un protocole de conversation dynamique.

La modélisation de protocole par automate d'état fini, permet de décrire le comportement des différentes entités(agents) impliquées dans une conversation. Avant chaque implémentation d'un protocole, il est nécessaire d'effectuer une validation de sa correction afin d'éviter les erreurs de communication pendant l'exécution du protocole.

Durant ce travail, nous avons étudié la modélisation de protocoles sous forme d'automate d'état fini. Pour avoir une meilleure modélisation, nous avons choisie le modèle de CAEFC, qui permet d'introduire de la modularité et de la hiérarchie sur les spécifications de service.
Pour valider de tel modèle, et après avoir étudié différentes techniques de validation qui ont étés proposés dans la littérature, nous avons choisie de suivre l'approche qui se base sur le principe de validation avec retour arrière. Notre choix est justifié par le fait que nous nous sommes intéressé au traitement des erreurs d'interblocage, qui sont facilement détectables par l'utilisation de cette approche. Pour rendre l'analyse plus efficace, nous avons optimisé l'opération de construction des chemins de retour arrière. Cette optimisation est obtenue par l'utilisation de la technique de construction des graphes de sauts(LRA). L'utilisation de LRA nous a permet d'éviter de parcourir des états qui ne sont pas utiles dans l'analyse, ce qui nous a permet de réduire considérablement la taille du graphe d'accessibilité.
Notre technique de validation permet de vérifier la présence ou l'absence des erreurs d'interblocage ainsi que la nature de chaque erreur détectée (simple, hybride ou complexe).

Les apports de notre technique peuvent être résumés comme suite:

- l'utilisation des automates d'états complexes permet d'avoir une vue de différents niveaux sur le service à modéliser, et donc une meilleure modélisation du service.

- Dans le cas où nous avons une spécification par automate d'état complexe stricte (chaque niveau de spécification converse avec le même niveau de spécification), chaque niveau de conversation sera validé indépendamment des autres.

- une approche de validation avec retour arrière, permet d'avoir une division automatique de l'opération de validation en plusieurs sous taches où chaque sous tache correspondra à un traitement d'un état suspect. Ce qui offre la possibilité d'effectuer plusieurs vérifications en parallèle.

- L'utilisation des graphes de sauts a permet de réduire considérablement la taille du graphe d'accessibilité de protocole.

Perspectives

Afin de compléter notre travail, les améliorations suivantes sont envisageables :

- Une extension sur les types d'erreur traitée, pour les transitions non exécutables et les réceptions non spécifiées.

- Pour rendre notre technique plus efficace, il est préférable d'effectuer une validation multi-niveaux, sans passer par l'aplatissement de la spécification du protocole. Avant de procéder avec ce type de validation un fondement théorique est nécessaire afin d'avoir une analyse correcte est complète du système vue la nature complexe et dynamique du modèle.

V) Références

[1] Brand, D., and Zafiropulo, P., "On communicating finite state machines", Journal of the ACM, April, 1983, volum30, no.2, pp.323-342.

[2] Alur, R., and Yannakakis, M., "Model checking of hierarchical state machines", ACM Transactions on Programming Languages and Systems(TOPLAS), 23(3), pp. 273-303, 2000.

[3] CACCIARI, L. and RAFIQ, O., "Decidability Issues in Reduced Reachability Analysis", T.A.S.C., Universitk de Pau, pp.158-165, 1993 IEEE.

[4] Cheung, S.C. and Kramer, J., "Enhancing Compositional Reachability Analysis with Context Constraints", Proc.ACM SIGSOFT"93: Symposium on the Foundations of Software Engineering, Los Angeles, California, December 1993.

[5] Choi, T.Y., "A structured approach to the analysis and design of finite state protocols", Ph.D.Thesis, School of Electrical Engineering, Georgia Institute of Technology, 1983.

[6] Choi, T.Y., and Miller, R.E. , "A decomposition method for the analysis and design of finite state protocols", in proceedings of Data Communication Symposium, ACM SIGCOMM, 1983, pp. 1048-1063.

[7] Booth, D., Haas, H., McCabe, F. and Newcomer, E.. Web services architecture, aout 2003. http ://www.w3.org/TR/2003/WD-ws-arch-20030808/.

[8] Gouda, M.G., and Yu, Y.T., "Protocol validation by maximal progress state exploration", in Proceedings of ACM SIGCOMM, 1983, pp. 68-75.

[9] GUITTON J. Planification multi-agent pour la composition dynamique de servicesWeb, Rapport de stage – Master 2 Recherche Intelligence Interaction Information, Universit´e Joseph Fourier (2006).

[10] Liu, H. and Miller, R.E. "Generalized Fair Reachability Analysis for Cyclic Protocols with Nondeterministic and Internal Transitions," ICNP'95, Tokyo, Nov. 7-10, 1995, pp. 6-13.

[11] Hans, V.S., and Hasan, U., "A uniform approach to tackle state explosion in verifying progress properties for networks of CFSMs*", TR-96-13, Department of Computer Science, University of Ottawa, November 1996.

[12] Holzmann, G.J., '' Design and validation of computer protocols '', Printice-Hall Inc., Englewood Cliffs, New Jersey, 1992.

[13] HONG, L. and RAYMOND, E.M."Partial-order validation for multi-process protocols modeled ascommunicating finite state machines", Bell Commun. Res., Morristown, NJ; pp. 76-83 ,29 Oct-1 Nov 1996.

[14] Hung, Y.C., and Chen, G.H., ''Reverse reachability analysis : a new technique for deadlock detection on communicating finite state machines", Softwar Practice and

Experience, September, 1993, volume 23(9), pp.88-93.

[15] Rubin, J. and Colin H. West. An improved protocol validation technique. Computer Networks: The international Journal of Distributed Informatique , 6(2):65-73, May 1982.

[16] Ozdemir, K., "Verifying the safety properties of concurrent systems via simultaneous reachability," Ph.D. Thesis, Department of CSI, University of Ottawa, 1995.

[17] Ozdemir, K. and Ural, H. "Protocol validation by simultaneous reachability analysis," *Computer Communications, 20(9): 772-788,* 1997.

[18] Lin, F.J., Chu, P.M. and Liu, M.T., "Protocol verification using reachability analysis: the state space explosion problem and relief strategies", Computer communications Review, 1987, volume 17, no.5, pp. 126-143.

[19] Gouda, M. G. and Ji-Yun Han. A finite state model for protocol processes and services. IEEE Transaction on Communication, COM-28:624-631, April 1980.

[20] Gouda, M.G. and Ji-Yun Han. Protocol validation by fair progress state exploration. Computer Networks and ISDN Systems, 9(5):353-361, 1985.

[21] Maria, C.Y., "survey of protocol verification techniques based on finite state machine models" CH2547-8/88/0000/0164 1988 IEEE.

[22] MELLITI T. Interopérabilité des services Web complexes. Application aux systèmes multi-agents, Thèse de Doctorat, Université Paris IX Dauphine(2004).

[23] Nicolle C.Définition des services-Web, Projet, Université Bourgogne (2002).

[24] Ozdemir, K., and Ural, H., "Deadlock detection in CFSM models via simultaneously executable sets", International Conference on Computing and Information, 1994, pp. 673-688.

[25] Zave , P., A Compositional Approach to Multiparadigm Programming. IEEE Software (1989), 15-25.

[26] Kellert, P. et Toumani, F. Les services Web sémantiques, Rapport de Recherche, LIMOS/RR 03-15, Juillet 2003.

[27] Peng, W., "Deadlock Detection in Communicating Finite State Machines by Even Reachability Analysis", Mobile Networks and Applications,Volume 2 , pp. 251 – 257,December 1997.

[28] Peng, W., and Purushothman, S., "Analysis of communicating processes for non progress", in Proceedings of IEEE conference on Distributed Computing Systems, June, 1989.

[29] Shing, C.H., and Jeff, K., "Context constraints for Compositional Reachability analysis" ACM transactions on software engineering and methodology, vol. 5; no. 4, October 1996.

[30] Tari, Z. and Arora, P., "A Communication Protocol Validation Approach based on Partial Exploration of Complex State Machines ",ICDCIT 2007.

[31] Tari, Z., and McKinlay, M., "DynWES A dynamic and interoperable protocol for Web services", International Symposium on Electronic Commerce (SEC), October 2002, pp. 74-86.

[32] Tari, Z., McKinlay, M., and Malhotra, M., "An XML-based conversational protocol for Web services" 18th ACM International Symposium on Applied Computing (SAC), May, 2003, pp., 1179-1148.

[33] VAN DER SCHOOT, H. and URAL, H., "A uniform approach to tackle state explosion in verifying progress properties for networks of CFSMs", TR-96-13, Department of Computer Science, University of Ottawa, November 1996.

[34] Vasudevan, V., 04 avril 2001.
http://webservices.xml.com/pub/a/2001/04/04/webservices/index.html.

[35]Peng, w. and Purushothaman, S., Data flow analysis of communicating finite state machines. ACM Transactions on Programming Languages and Systems, Vol. 13, No. 3, July 1991, pp.399-442.

[36] West, C.H., "Protocol validation by random state exploration", Protocol Specification, Testing and Verification, VI. North Holland, 1987, pp.233-242.

[37] YU, L. and KUO-CHUNG, T., "Blocking-based Simultaneous Reachability Analysisof Asynchronous Message-passingPrograms", Proceedings of the 13 th International Symposium on Software Reliability Engineering (ISSRE'02), 2002 IEEE.

[38] Tari, Z., Malhotra, M., TARI, A. et McKINLAY, M.: *Toward the right communication protocole for web services*. International Journal of Web Service Research, 2(2): 19-42 (2005).

[39] Heather, K., Web services conceptual architecture (wsca 1.0), may 2001.
http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.

[40] Brams G. W., Théorie et pratiques. Volume 1 et 2, Masson, Paris, 1983.

Annexes

1) Pseudo code de l'algorithme RLRA

La première phase de l'algorithme, qui assure la détection des états suspects d'interblocage, est implémentée par la procédure DES(Détection des état Suspects). La procédure reçoit en entrée les paramètres suivants :

P : un ensemble d'AEFC communicants du protocole :(M1, M2).

Chaque AEFC Mi (i=1,2) est défini par un quintuple **(E_i,i_i,E_{fi},A_i,T_i) :**

E_i : l'ensemble de tous les états de l'entité i. Chaque état **$e \in E_i$** est défini par les informations suivantes :(**nom-état** : String, **Est-Final** :Boulen, **Est-Complexe** :Boulean).

T_i : l'ensemble de toutes les transitions de l'AEFC Mi. Une transition est définie par les informations suivantes : (debut_transition :$e \in E_i$,événement :enum(+,-), message :$m \in A_i$, fin-transition : $e \in E_i$).

A la fin de l'exécution, la procédure renvoi le résultat suivant :

Etat_Suspect: l'ensemble des états globaux suspects d'interblocage. Chaque état global est sous la forme : (etat1 :$e \in E_1$,etat2 :$e \in E_2$,Canal12 :$msg \in A_1$, Canal21 :$msg \in A_2$).

Procédure de détection des états globaux suspects :

```
Procedure DES (P, Etat_Suspect)

    Foreach e1∈E1 and Emission(e1,T1) do

        Foreach e2∈E2 and Emission(e2,T2) do

            If non(e1.Est-Final and e2.Est-Final ) then
            Etat_Suspect = Etat_Suspect ∪ ( e1,e2,∅, ∅).
            fi ;
        od ;
od ;


Fonction Emission(Etat :e ,T :T1)

        Foreach t∈T1 and t.debut_transition =e do

                If t.évènement='-' return Faux;

    Return Vrai;
```

La confirmation des erreurs logiques du protocole est assurée par la fonction RLRA().

Les paramètres d'entrés de la fonction sont :

EG : Un état global du protocole, qui est composé des éléments suivants :
(e1 :Etat entité1,e2 :Etat entité2, C12 :contenu du canal de communication C12,
C21 :contenu du canal de communication C21).
Chaque état $\mathbf{e_i \in E_i}$ (i={1,2}) est défini par les informations suivantes :(**nom-état** : String, **Est-Final** :Boulen, **Est-Complexe** :Boulean).
$\mathbf{T_i}$: l'ensemble de toutes les transitions de l'AEFC Mi (i={1,2}). Une transition est définie par les informations suivantes : (debut_transition :e∈E_i,événement :enum(+,-), message :m∈A_i, fin-transition : e∈E_i).

En sortie, la fonction renvoie un boulean :
Boolean : prend la valeur vraie si l'état global est confirmé comme étant une erreur logique, sinon il prend la valeur faux.

Fonction de confirmation des erreurs d'interblocage :

```
fonction RLRA(EG,T1,T2,EG0,Chemin) :Boulean

  if EG=EG0 then  //condition d'arrêt de l'analyse(état initial atteint)
  return true ;
  fi ;

  /*** Calcul de l'ensemble Wait ***/
         If  P^R_1(EG)≠∅
            Wait←Wait∪indice(M1);
           fi;
         If  P^R_2(EG)≠∅
            Wait←Wait∪indice(M2);
           fi;
  /*** Fin Calcul Wait ***/


 /***Traitement cas Wait=I; pas de transition à exécuter simultanément***/
 if  Wait={1,2} then
     Foreach t∈ X_1^R(EG) do

        If  t∈ X_1^R-(EG)  then
            EG.e1←t.debut_transition  ; E.C12←C12- t.message;
        fi ;
       else  EG.e1←t.debut_transition  ; EG.C21←C21+ t.message;

          If (EG∉ DejaTraite) then
                DejaTraite← DejaTraite∪EG ;
                If(RLRA(EG,T1,T2,EG0)) return true ;
        fi ;
    od ;

     Foreach t∈ X_2^R(EG) do

        If  t∈ X_2^R-(EG)  then
            EG.e2←t.debut_transition  ; EG.C21←C21- t.message;
       fi ;
       else  EG.e2←t.debut_transition  ; EG.C12←C12+ t.message;

       If (EG∉ DejaTraite) then
```

```
                DejaTraite← DejaTraite∪EG ;
                If(RLRA(EG,T1,T2,EG0)) return true ;
        fi ;
    od ;
fi;/***Fin Wait=I***/
else /***  wait⊂I; une des deux entité dois être bloquée ***/

   if wait={1} then  //on bloque l'entité M1, seuls les transitions de M2 seront exécutée
     Foreach t∈ X2R(EG) do
        If  t∈ X2R-(EG)  then
            EG.e2←t.debut_transition  ; EG.C21←C21- t.message;
       fi ;
       else  EG.e2←t.debut_transition  ; EG.C12←C12+ t.message;
       If (EG∉ DejaTraite) then
                DejaTraite← DejaTraite∪EG ;
                If(RLRA(EG,T1,T2,EG0)) return true ;
        fi ;
    od ;
  fi ;
  else  if  wait={2} then //on bloque l'entité M2, seuls les transitions de M1 seront exécutée
        Foreach t∈ X1R(EG) do

        If  t∈ X1R-(EG)  then
            EG.e1←t.debut_transition  ; E.C12←C12- t.message;
        fi ;
        else  EG.e1←t.debut_transition  ; EG.C21←C21+ t.message;

          If (EG∉ DejaTraite) then
                DejaTraite← DejaTraite∪EG ;
                If(RLRA(EG,T1,T2,EG0)) return true ;
          fi ;
    od ;
  fi ;
else   //wait=∅; toutes les transitions exécutables des deux entités seront exécutes simultanément
      Foreach t1∈ X1R(EG) do
          Foreach t2∈ X2R(EG) do

            If  t1∈ X1R-(EG) then
                EG.e1←t1.debut_transition  ; EG.C12←C12- t.message ;
           fi ;
            else EG.e1←t.debut_transition  ; EG.C21←C21+ t.message;
            fi;
              If  t2∈ X2R-(EG) then
                EG.e2←t2.debut_transition  ; EG.C21←C21- t.message ;
               fi ;
            else EG.e2←t2.debut_transition  ; EG.C12←C12+ t2.message;fi;

               If (EG∉ DejaTraite) then
                DejaTraite← DejaTraite∪EG ;
                If(RLRA(EG,T1,T2,EG0)) return true ;
            fi ;

          od ;
      od ;
  fi ;
  return false ;
```

2) Fichiers de specification XML

Spécification du CAEF de la Figure 5.4

```
<?xml version="1.0" encoding="UTF-8"?>
<CAEF>
<Automate nom="M1"><!-- ********************M1********************-->
<Etat nom="0" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>a</Message>
<Vers>1</Vers>
</Transition>
</Etat>
<Etat nom="1" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>b</Message>
<Vers>2</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>c</Message>
<Vers>3</Vers>
</Transition>
</Etat>
<Etat nom="2" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>d</Message>
<Vers>4</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>c</Message>
<Vers>0</Vers>
</Transition>
</Etat>
<Etat nom="3" EstFinale="vrai" EstComplexe="faux">
</Etat>
<Etat nom="4" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>b</Message>
<Vers>2</Vers>
</Transition>
</Etat>
</Automate><!-- ******************************Fin M1
****************************************-->
<Automate nom="M2"> <!--********************M2********************-->
<Etat nom="0" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>a</Message>
<Vers>1</Vers>
</Transition>
</Etat>
<Etat nom="1" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
```

```
<Message>b</Message>
<Vers>2</Vers>
</Transition>
</Etat>
<Etat nom="2" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>e</Message>
<Vers>0</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>d</Message>
<Vers>3</Vers>
</Transition>
</Etat>
<Etat nom="3" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>b</Message>
<Vers>2</Vers>
</Transition>
</Etat>
</Automate><!-- ********************Fin M2********************-->
</CAEF>
```

Spécification du CAEF de la Figure 5.7

```
<?xml version="1.0" encoding="UTF-8"?>
<CAEF>
<Automate nom="M1"><!-- ********************M1********************-->
<Etat nom="0" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>a</Message>
<Vers>1</Vers>
</Transition>
</Etat>
<Etat nom="1" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>b</Message>
<Vers>2</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>c</Message>
<Vers>3</Vers>
</Transition>
</Etat>
<Etat nom="2" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>d</Message>
<Vers>4</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>a</Message>
<Vers>1</Vers>
```

```
</Transition>
</Etat>
<Etat nom="3" EstFinale="vrai" EstComplexe="faux">
</Etat>
<Etat nom="4" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>+</Evenement>
<Message>e</Message>
<Vers>5</Vers>
</Transition>
</Etat>
<Etat nom="5" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>+</Evenement>
<Message>f</Message>
<Vers>4</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>h</Message>
<Vers>7</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>g</Message>
<Vers>6</Vers>
</Transition>
</Etat>
<Etat nom="6" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>+</Evenement>
<Message>i</Message>
<Vers>6</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>j</Message>
<Vers>8</Vers>
</Transition>
</Etat>
<Etat nom="7" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>-</Evenement>
<Message>d</Message>
<Vers>4</Vers>
</Transition>
</Etat>
<Etat nom="8" EstFinale="vrai" EstComplexe="faux">
</Etat>
</Automate><!-- ******************************Fin M1
****************************************-->
<Automate nom="M2"> <!--********************M2******************-->
<Etat nom="0" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>a</Message>
<Vers>1</Vers>
</Transition>
</Etat>
<Etat nom="1" EstFinale="faux" EstComplexe="faux">
<Transition>
```

<Evenement>+</Evenement>
<Message>b</Message>
<Vers>2</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>c</Message>
<Vers>0</Vers>
</Transition>
</Etat>
<Etat nom="2" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>d</Message>
<Vers>3</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>a</Message>
<Vers>1</Vers>
</Transition>
</Etat>
<Etat nom="3" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>e</Message>
<Vers>4</Vers>
</Transition>
</Etat>
<Etat nom="4" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>h</Message>
<Vers>2</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>f</Message>
<Vers>3</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>g</Message>
<Vers>5</Vers>
</Transition>
</Etat>
<Etat nom="4" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>i</Message>
<Vers>5</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>j</Message>
<Vers>6</Vers>
</Transition>
</Etat>
<Etat nom="6" EstFinale="vrai" EstComplexe="faux">
</Etat>
</Automate><!-- **********Fin M2**********-->

```
</CAEF>
```

Spécification du CAEF de la Figure 5.10

```
<?xml version="1.0" encoding="UTF-8"?>

<CAEF>
<Automate nom="M1"><!-- ********************M1******************-->
<Etat nom="0" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>a</Message>
<Vers>1</Vers>
</Transition>
</Etat>
<Etat nom="1" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>b</Message>
<Vers>2</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>c</Message>
<Vers>3</Vers>
</Transition>
</Etat>
<Etat nom="2" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>d</Message>
<Vers>4</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>a</Message>
<Vers>1</Vers>
</Transition>
</Etat>
<Etat nom="3" EstFinale="vrai" EstComplexe="faux">
</Etat>
<Etat nom="4" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>-</Evenement>
<Message>e</Message>
<Vers>5</Vers>
</Transition>
</Etat>
<Etat nom="5" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>+</Evenement>
<Message>f</Message>
<Vers>4</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>h</Message>
<Vers>7</Vers>
</Transition>
<Transition>
```

<Evenement>+</Evenement>
<Message>g</Message>
<Vers>6</Vers>
</Transition>
</Etat>
<Etat nom="6" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>+</Evenement>
<Message>i</Message>
<Vers>6</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>j</Message>
<Vers>8</Vers>
</Transition>
</Etat>
<Etat nom="7" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>-</Evenement>
<Message>d</Message>
<Vers>4</Vers>
</Transition>
</Etat>
<Etat nom="8" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>-</Evenement>
<Message>k</Message>
<Vers>9</Vers>
</Transition>
</Etat>
<Etat nom="9" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>+</Evenement>
<Message>l</Message>
<Vers>10</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>m</Message>
<Vers>11</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>n</Message>
<Vers>11</Vers>
</Transition>
</Etat>
<Etat nom="10" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>-</Evenement>
<Message>o</Message>
<Vers>8</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>m</Message>
<Vers>11</Vers>
</Transition>
</Etat>
<Etat nom="11" EstFinale="faux" EstComplexe="vrai">

```
<Transition>
<Evenement>+</Evenement>
<Message>q</Message>
<Vers>12</Vers>
</Transition>
</Etat>
<Etat nom="12" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>+</Evenement>
<Message>r</Message>
<Vers>12</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>p</Message>
<Vers>13</Vers>
</Transition>
</Etat>
<Etat nom="13" EstFinale="vrai" EstComplexe="faux">
</Etat>
</Automate><!-- ******************************Fin M1
****************************************-->
<Automate nom="M2"> <!--********************M2********************-->
<Etat nom="0" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>a</Message>
<Vers>1</Vers>
</Transition>
</Etat>
<Etat nom="1" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>b</Message>
<Vers>2</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>c</Message>
<Vers>0</Vers>
</Transition>
</Etat>
<Etat nom="2" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>d</Message>
<Vers>3</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>a</Message>
<Vers>1</Vers>
</Transition>
</Etat>
<Etat nom="3" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>+</Evenement>
<Message>e</Message>
<Vers>4</Vers>
</Transition>
</Etat>
```

<Etat nom="4" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>h</Message>
<Vers>2</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>f</Message>
<Vers>3</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>g</Message>
<Vers>5</Vers>
</Transition>
</Etat>
<Etat nom="5" EstFinale="faux" EstComplexe="faux">
<Transition>
<Evenement>-</Evenement>
<Message>i</Message>
<Vers>5</Vers>
</Transition>
<Transition>
<Evenement>+</Evenement>
<Message>j</Message>
<Vers>6</Vers>
</Transition>
</Etat>
<Etat nom="6" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>+</Evenement>
<Message>k</Message>
<Vers>7</Vers>
</Transition>
</Etat>
<Etat nom="7" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>+</Evenement>
<Message>m</Message>
<Vers>8</Vers>
</Transition>
<Transition>
<Evenement>-</Evenement>
<Message>n</Message>
<Vers>8</Vers>
</Transition>
</Etat>
<Etat nom="8" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>+</Evenement>
<Message>q</Message>
<Vers>9</Vers>
</Transition>
</Etat>
<Etat nom="9" EstFinale="faux" EstComplexe="vrai">
<Transition>
<Evenement>-</Evenement>
<Message>r</Message>
<Vers>9</Vers>
</Transition>

<Transition>
<Evenement>+</Evenement>
<Message>p</Message>
<Vers>10</Vers>
</Transition>
</Etat>
<Etat nom="10" EstFinale="vrai" EstComplexe="faux">
</Etat>
</Automate><!-- **********Fin M2**********-->
</CAEF>